Technische Universität München
Fakultät für Elektro- und Informationstechnik
Lehrstuhl für Sicherheit in der Informationstechnik

Reverse Engineering of Synthesised Netlists

Master Thesis

Johanna Baehr

Technische Universität München
Fakultät für Elektro- und Informationstechnik
Lehrstuhl für Sicherheit in der Informationstechnik

Reverse Engineering of Synthesised Netlists

Master Thesis

Johanna Baehr

Betreuer:	Michael Tempelmeier M.Sc.
Betreuender Hochschullehrer:	Prof. Dr.-Ing. Georg Sigl
Ausgabe des Themas:	July 30th, 2015
Abgabedatum:	March 21st, 2016

Johanna Baehr
Arcisstraße 21
80333 München

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht.

München, den 21. März 2016

Johanna Baehr

Contents

1 Motivation **1**

2 Formal Defintion **3**
2.1 Basic Defintions . . . 3
2.1.1 Graph Theory . . . 3
2.1.2 Netlist Definition . . . 3
2.2 Problem Definition . . . 4
2.3 Solution Definition . . . 4
2.3.1 Candidate Subcircuit Enumeration . . . 4
2.3.2 Subcircuit Identification . . . 5

3 Related Work **7**

4 Methodology **12**
4.1 Candidate Subcircuit Enumeration . . . 13
4.1.1 Word Identification . . . 13
4.1.2 Data Path Identification . . . 22
4.1.3 Candidate Subcircuit Generation . . . 25
4.2 Circuit Clustering . . . 27
4.3 Subcircuit Identification . . . 29
4.3.1 Component Library creation . . . 30
4.3.2 Input/Output Correspondence . . . 31
4.3.3 Satisfiability solving . . . 33
4.3.4 Quantified Boolean Formula Satisfiability Solving . . . 35
4.3.5 Satisfiability modulo Theories solving . . . 37
4.4 Post Processing . . . 39

5 Results **40**
5.1 Candidate Subcircuit Enumeration . . . 40
5.2 Circuit Clustering . . . 44
5.3 Subcircuit Identification . . . 46

6 Conclusion **49**

Acronyms **50**

Bibliography **52**

List of Figures

4.1 Design Flow 12
4.2 Structural Word Identification 14
4.3 Node Structure for feasible cut calculation 16
4.4 Permutation Coefficents for three Inputs Function 20
4.5 Identification of Chain Words 20
4.6 Identification of Input Words 21
4.7 Symbolic Evaluation for Word Propagation 24
4.8 Satisfiability (SAT) Solving 34
4.9 Quantified Boolean Formula (QBF) Solving 36

5.1 Cluster Example of IDEA round module using MCL Algorithm 45
5.2 Cluster Example of S-Box using MCL Algorithm 45
5.3 Cluster Example of IDEA round module using Louvain Method 46
5.4 Cluster Example of S-Box using Louvain Method 46
5.5 Runtime comparison 47

List of Tables

4.1 Precomputed Coefficients for three Input Function 19
4.2 Re-defintion of XOR cell for symbolic evaluation 23

5.1 Design Information 41
5.2 Word Identification 41
5.3 Word Propagation 42
5.4 Subcircuit Generation 43
5.5 Clustering Data 44
5.6 Subcircuit Identification 47

List of algorithms

1 Calculate set of k-feasible cuts 16
2 Markov Cluster Algorithm 28

Chapter 1: Motivation

Over the past years, the trend in hardware development has gone towards Third Party Intellectual Property (3PIP) Cores and commercial off-the-shelf Integrated Circuits (ICs) [47], as high-level design is being outsourced to foundries [6]. This gives way to a number of security threats, such as the insertion of Hardware Trojans (HTs), or theft of Intellectual Property (IP) through illegal reverse engineering, and so countermeasures must be made available.

Different methods for the detection of malicious hardware within 3PIP Cores exist. Comparison of functional or timing behaviour of ICs can can detect possible threats, but this requires the existence of a golden model [6]. In particular with the increasing availability of post-synthesis hardware, without the corresponding Register Transfer Level (RTL) level descriptions, the detection of malicious hardware code has become more and more important in the last years [42, 43]. Reverse Engineering can provide a convenient tool to facilitate threat identification, by creating a better understanding of the unknown synthesised circuit. In the past, reverse engineering was a mostly manual task, often carried out by a hardware engineer by inspection, but, with more focus placed on this topic, it is becoming increasingly automated. The purpose of reverse engineering unstructured netlists is, now as then, to establish a higher level description of the functionality of the circuit. Possible areas which may constitute a threat can then be identified and further examined.

On the other hand, illegal reverse engineering of IP causes a significant financial cost to the hardware industry. With the development of new methods to reverse engineering the physical structure of ICs, high-level reverse engineering is becoming an increasingly employed tool in design piracy. To protect IP from theft, countermeasures are required. Furthermore, particularly in the field of cryptology, reverse engineering can severely impact the security of encryption and decryption algorithms. To protect the integrity of the design, obfuscation, both on a physical and netlist level, is becoming more and more prevalent. Understanding the process behind reverse engineering can provide insights into future possibilities for obfuscation or other countermeasures. Additionally, a reverse engineering framework can be used to verify, whether possible countermeasures are successful in preventing extraction of design functionality.

The primary challenge of reverse engineering an unknown netlist lies in the complete lack of structure within the synthesised netlist, and the vast range of possibilities of functionality and implementation of the design [43]. While small designs may contain the same types of functional cells with a similar structure, independent of synthesis tool or optimisation, larger, composite designs will by synthesised differently depending on cell library, design tool, and design implementation. As no information is given

regarding the original functionality of the design, the search space for functionality and implementation is enormous.

This thesis tests the viability of a method to reverse engineer gate level designs, building on previous works on this topic [43, 65]. In chapter 2, the reverse engineering problem will be formally defined, including a set of basic definitions to give the reader a better understanding of the subsequent chapters. Many of the employed ideas originate from the area of design verification and equivalence checking. A brief overview of previous works, not only on the topic of reverse engineering, but also the methods used in this thesis are discussed in chapter 3. A detailed view of the methodology of reverse engineering is described in chapter 4, with a strong focus on implementation and algorithms, followed by a discussion of the results in chapter 5. Finally, a conclusion, including a recommendation for further research topics, is presented in chapter 6.

Chapter 2: Formal Defintion

2.1 Basic Defintions

2.1.1 Graph Theory

In this chapter, a brief introduction to the basic definitions and concepts used in this paper is presented. A synthesised netlist $\mathcal{N}$ can be represented as a directed acylic graph (DAG) $\mathcal{G}$, where a *node* n of $\mathcal{G}$ is equivalent to a functional cell in $\mathcal{N}$, and a directed *edge* e (u, v) in $\mathcal{G}$ represents a connection between two functional cells in $\mathcal{N}$. For this DAG $\mathcal{G}$, the following is defined:

Definition 1. *The* ***path*** $(u \rightarrow v)$ *is a set of nodes connecting* (u, v)*, with* u *as the tail and* v *as the head.*

Definition 2. *A node* u *is called an* ***ancestor*** *of node* v*, if there exists a path* $(u \rightarrow v)$*. A node* u *is called a* ***predecessor*** *of node* v*, if there exists an edge* (u, v)*.*

Definition 3. *A node* v *is called a* ***descendant*** *of node* u*, if there exists a path* $(u \rightarrow v)$*. A node* v *is called a* ***successor*** *of node* u*, if there exists an edge* (u, v)*.*

Definition 4. *The* ***distance*** *between nodes* (u, v) *is the length of the shortest path between* (u, v)*. If no path exists, the distance is infinite.*

2.1.2 Netlist Definition

A synthesised combinational netlist $\mathcal{N}$ can be represented as a tuple $(\mathcal{I}, \mathcal{O}, \mathcal{F}, \mathcal{G}, \mathcal{W})$, where:

- Netlist $\mathcal{N}$ can be represented as a DAG $\mathcal{G}$
- $\mathcal{I} = \{i_k \mid i_k \in \{0, 1\}, k \in \mathbb{N}\}$ is a finite set of primary input signals, with an optional set of groupings of input signals $\overrightarrow{\mathcal{I}} = \{\overrightarrow{i_k} \mid \overrightarrow{i_k} \subseteq \mathcal{I}, k \in \mathbb{N}\}$
- $\mathcal{O} = \{o_k \mid o_k \in \{0, 1\}, k \in \mathbb{N}\}$ is a finite set of primary output signals, with an optional set of groupings of input signals $\overrightarrow{\mathcal{O}} = \{\overrightarrow{o_k} \mid \overrightarrow{o_k} \subseteq \mathcal{O}, k \in \mathbb{N}\}$
- $\mathcal{F} : \mathcal{I} \rightarrow \mathcal{O}$ is a set of logic formula describing the input-output behaviour of the netlist
- $\mathcal{G} = \{g_k \mid k \in \mathbb{N}\}$ is a set of combinational gates, where

- $\mathcal{W} = \{w_k \mid k \in \mathbb{N}\}$ is a set of internal wires interconnecting gates in $g_k \in \mathcal{G}$
- each gate $g_k \in \mathcal{G}$ is associated with gate-type according to the cell-library
- each gate $g_k \in \mathcal{G}$ is associated with a set of gate inputs $I_g = \{w_k \mid w_k \in \mathcal{W}, k \in \mathbb{N}\}$, a set of gate outputs $O_g = \{w_k \mid w_k \in \mathcal{W}, k \in \mathbb{N}\}$ where $I_g \cap O_g = \emptyset$, and a boolean function $F_g : I_g \rightarrow O_g$

2.2 Problem Definition

The problem posed by the reverse engineering of synthesised netlists is to restore a high level, functional description of a gate-level netlist. This can be formulated into two distinct problems:

1. *Candidate Subcircuit Enumeration,* which consists of the detection of a set of potential functional modules $\mathcal{C} = \{c_k \mid c_k \subseteq \mathcal{G}, k \in \mathbb{N}\}$ within the synthesised netlist $\mathcal{N}$.

2. *Subcircuit Identification,* which seeks to identify the functionality of a candidate subcircuit c_k, through comparison with a pre-defined component library using Combinational Equivalence Checking (CEC)

Once a set of functional submodules is identified, the submodules can be combined to create a functional description of the netlist.

2.3 Solution Definition

2.3.1 Candidate Subcircuit Enumeration

Before synthesis, most circuits consist of a set of interconnected modules, of which each implements a specific function. During synthesis, these functional blocks are integrated into a single netlist $\mathcal{N}$, describing the functionality of the entire circuit. In order to extract these functional modules or candidate subcircuits, multiple steps are required:

- Given an unknown netlist $\mathcal{N}$, containing gates $g_k \in \mathcal{G}$ and interconnected by wires $w_k \in \mathcal{W}$, a set of unordered word level structures $\overrightarrow{\mathcal{W}} = \{\overrightarrow{w_k} \mid \overrightarrow{w_k} \subseteq \mathcal{W}, k \in \mathbb{N}\}$ can be extracted.

- Netlist $\mathcal{N}$ is associated with a data path $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, containing a set of vertices $\mathcal{V} = \overrightarrow{\mathcal{W}}$ and a set of edges $\mathcal{E} \subseteq \{\{u, v\} \mid u, v \in \mathcal{V}\}$, which describes the relation between word level structures within the netlist $\mathcal{N}$.

- A set of candidate functional modules or subcircuits $\mathcal{C} = \{c_k \mid c_k \subseteq \mathcal{G}, k \in \mathbb{N}\}$ are generated by carving cells $\mathcal{G}_c \subseteq \mathcal{G}$ between connected words $\overrightarrow{\mathcal{W}}$ of the data path $\mathcal{D}$. Each candidate subcircuit c_k is associated with:
  - a set of subcircuit inputs $\mathcal{X} = \{x_k \mid x_k \in \mathcal{I} \cup \mathcal{W}, k \in \mathbb{N}\}$
  - a set of subcircuit outputs $\mathcal{Y} = \{y_k \mid y_k \in \mathcal{O} \cup \mathcal{W}, k \in \mathbb{N}\}$

  - input-output behaviour $\mathcal{Z} : \mathcal{X} \rightarrow \mathcal{Y}$
  - a set of combinational gates $\mathcal{G}_c \subseteq \mathcal{G}$
  - a set of internal wires $\mathcal{W}_c \subseteq \mathcal{W}$

- A set of circuit partitons $\mathcal{P} = \{p_k \mid p_k \subset \mathcal{G}, k \in \mathbb{N}\}$ can provide further information regarding the boundaries of functional modules. It is possible that these circuit partitions are overlapping.

2.3.2 Subcircuit Identification

Once a set of candidate subcircuits has been generated, Combinational Equivalence Checking (CEC) can be used to identify the functionality of the subcircuit. Given an unknown netlist $\mathcal{N}$, and an abstract component library $\mathcal{L}$, the problem consists of identifying the functionality of all previously determined subcircuits in $\mathcal{C}$. For each $c_k \in \mathcal{C}$, this can be done by identifying an abstract component within a pre-defined component library $\alpha \subset \mathcal{L}$ such that both are equivalent: $c_k \equiv \alpha$.

Again, a set of steps is required to generate the necessary information for the CEC Problem:

- A library of abstract components $\mathcal{L} = \{\alpha_k \mid k \in \mathbb{N}\}$ is created, against which candidate subcircuits can be matched. Each component α_k is associated with :
  - a finite set of primary input signals $I = \{i_k \mid i_k \in \{0, 1\}, k \in \mathbb{N}\}$
  - a finite set of output signals $O = \{o_k \mid o_k \in \{0, 1\}, k \in \mathbb{N}\}$
  - a formal description demonstrating input-output behaviour $\mathcal{S} : I \rightarrow O$

- A set of possible Input/Output (I/O) permutations must be determined to solve the CEC Problem. Given a subcircuit $c_k \in \mathcal{C}$ with a set of Inputs $\mathcal{X}$, a set of Outputs $\mathcal{Y}$, and with functionality $\mathcal{Z} = (z_1(\mathcal{X}), z_2(\mathcal{X}), \ldots, z_{|\mathcal{Y}|}(\mathcal{X}))$, and an abstract component $\alpha \in \mathcal{L}$ with a set of Inputs I, set of Outputs O, and functionality $\mathcal{S} = (s_1(I), s_2(I), \ldots, s_{|O|}(I))$, a set of all possible input permutations $\mathcal{P}_i = (\pi_1, \pi_2, \ldots)$ and output permutations $\mathcal{P}_o = (\phi_1, \phi_2, \ldots)$ must be found, such that:

$$\exists \pi \in \mathcal{P}_i,\ \exists \phi \in \mathcal{P}_o,\ \forall z_i \in \mathcal{Z},\ \forall s_i \in \mathcal{S}\ :\ s_{\phi(i)}(\mathcal{I}) = z_i(\pi(X)) \tag{1}$$

There exists a multitude of solutions to the CEC Problem, in this thesis, Satisfiability (SAT) Solving, Quantified Boolean Formula (QBF) Solving and a Satisfiability module Theories (SMT) based approach using templating are implemented and compared.

2.3.2.1 Satisfiability (SAT) Solving

SAT based equivalence solving seeks to prove the functional equivalence between two circuits, by testing whether the boolean function created by the miter circuit of the two circuits is satisfiable.

Given subcircuit $c \in \mathcal{C}$ with functionality $\mathcal{Z} = z_1(\mathcal{X}), z_2(\mathcal{X}), \ldots, z_{|\mathcal{Y}|}(\mathcal{X})$ and abstract component $\alpha \in \mathcal{L}$ with functionality $\mathcal{S} = s_1(I), s_2(I), \ldots, s_{|O|}(I)$, under the

assumption that $|\mathcal{X}| = |I|,\ |\mathcal{Y}| = |O|$, the equivalence between the subcircuit and abstract component is given, only if the boolean function of the the miter circuit is not satisfiable:

$$c \equiv \alpha \iff (z_1(\mathcal{X}) \oplus s_1(I)) \vee \cdots \vee (z_{|\mathcal{Y}|}(\mathcal{X}) \oplus s_{|O|}(I)) = 1 \text{ is not satisfiable} \tag{2}$$

As all inputs are existentially quantified, the miter circuit seeks to disprove equivalence. The theory of SAT Solving is further discussed in subsection 4.3.3.

2.3.2.2 Quantified Boolean Formula Satisfiability Solving

Quantified Boolean Formula (QBF) Solving provides an extension to SAT Solving, allowing for the universal or existential quantification of variables. For the identification of a candidate subcircuit, control wires as side inputs are possible. Thus the functionality of the candidate subcircuit does not need to match that of the component completely, except under a specific assignment of control wires. Similarly to SAT solving, a quantified boolean formula is created with a miter circuit and tested for satisfiability, but the miter circuit is constructed to prove equivalence.

Given subcircuit $c \in \mathcal{C}$ with set of inputs $\mathcal{X}$, set of side Inputs $\mathcal{X}_s$, and with functionality $\mathcal{Z} = z_1(\mathcal{X} \cup \mathcal{X}_s), z_2(\mathcal{X} \cup \mathcal{X}_s), \ldots, z_{|\mathcal{Y}|}(\mathcal{X} \cup \mathcal{X}_s)$, and given abstract component $\alpha \in \mathcal{L}$ with functionality $\mathcal{S} = s_1(I), s_2(I), \ldots, s_{|O|}(I)$, under the assumption that $|\mathcal{X}| = |I|,\ |\mathcal{Y}| = |O|$, the equivalence between a part of the subcircuit $c_i \subset c$ and the abstract component α is given only if the quantified boolean function of the miter circuit is satisfiable:

$$\begin{aligned} &c_i \equiv \alpha, c_i \subseteq c \iff \\ &\exists \mathcal{X}_s,\ \forall \mathcal{X}\ \overline{(z_1(\mathcal{X} \cup \mathcal{X}_s) \oplus s_1(I))} \wedge \cdots \wedge \overline{(z_{|\mathcal{Y}|}(\mathcal{X} \cup \mathcal{X}_s) \oplus s_{|O|}(I))} = 1 \text{ is satsifiable} \end{aligned} \tag{3}$$

In subsection 4.3.4, further information regarding QBF based equivalence solving will be detailed.

2.3.2.3 Satisfiability modulo Theories solving

SMT based CEC extends traditional SAT solving to other domain theories outside of the boolean domain [74]. Instead of creating a boolean formula, other theories such as bit vectors, arrays or arithmetic can be used to specify the component. In order to extend the implementation space of such a component, a template based approach is used.

The functionality $\mathcal{Z}$ of the candidate subcircuit c_k is equivalent to that of functionality $\mathcal{S}$ of a template $\mathcal{T}$, if c_k can be mapped to the template, such that a SMT instance $\mathcal{T}_{c_k}$ is created, and this instance is satisfiable for all data inputs $\mathcal{I}$, and for a given assignment of control inputs $\mathcal{I}_c$:

$$\mathcal{Z} \equiv \mathcal{S} \iff \exists \mathcal{I}_c\ \forall \mathcal{I}\ \mathcal{T}_{c_k} : c_k \mapsto \mathcal{T} \text{ is satisfiable} \tag{4}$$

Subsection 4.3.5 will provide further information regarding the SMT based solving, and how templating is used to increase the design space of each component.

Chapter 3: Related Work

Reverse Engineering of synthesised netlist has become more common in literature within the past ten years. The first major work on netlist reverse engineering was presented by Hansen et al., [35] in 1999, who successfully reverse engineering the ISCAS-85 Combinational Benchmark Circuits using manual strategies. While this was a first step towards better understanding the reverse engineering process, further work towards the automation of reverse engineering was not presented until much later. Perhaps the most comprehensive approach to automated combinational and sequential reverse engineering is provided by Subramanyan et al. [65], who combines the ideas of previous literature [41, 43, 67] to present a complete framework for re-establishing structure in synthesised netlist.

To understand the context of reverse engineering within hardware security, the development of hardware threats must be considered. A comprehensive overview of current issues in hardware security is presented by Rostami et al. [56], listing both Hardware Trojans (HTs) in 3PIP and reverse engineering of Intellectual Property (IP) or ICs as substantial threats. A number of methods to prevent and detect HTs in gate level netlists exist, for example, through error detection techniques [78], signal correlation based clustering [13], score-based classification [53], or signal based methods [4]. Furthermore, a method based on reverse engineering, through the creation of a golden model, is detailed in [5, 6]. The method proposed in this thesis is complementary to these efforts, as it is not designed to identify HTs, but rather to give high-level, functional model of the design, simplifying future HT detection.

With the availability of more methods for reverse engineering of the physical structure of ICs, such as detailed in [36, 46, 68], reverse engineering has become a more significant threat to the chip industry. Not only can IP be reproduced illegally, but in particularity in the field of cryptology, reverse engineering of ICs can severely impact the security of Encryption Algorithms, as demonstrated in [66]. One of the most commonly use countermeasure against both malicious hardware and illegal reverse engineering is design obfuscation. This can be done both on the physical design and the gate level netlist [56], for example through the use of permutation blocks [33]. McLoughlin [46] presents a number of additional passive and active reverse engineering mitigation methods. Through understanding the concepts and possibilities of automated reverse engineering, in the future, better methods for design obfuscation may become available.

Subcircuit Enumeration

Current research in netlist reverse engineering suggest that the problem can be

divided into two parts, starting with the extraction of candidate subcircuits from the gate-level netlist [43, 65]. Li et al. [43] presents the first functional approach to automatically extract functional modules from gate-level netlists, by first identifying the flow of data or data path within the circuit, and carving out functional modules between parts of the data path.

In the past, a number of approaches to extract small subcircuits from a gate-level netlist were presented, mostly applied to the fields of circuit verification and design automation. Here the size of extracted circuits does not exceed a given number of gates (n > 20). An early approach to regularity extraction in data paths is given by Chowdhary et al., [16], who propose to use templates to extract a set of functionally equivalent set of subcircuits from a given netlist. SubIslands [58] and SubHunter [64] discuss further solutions to this problem. White et al. [75, 76] first pose the candidate subcircuit enumeration problem, which extracts candidate subcircuits which correspond to high-level functional modules. They propose an exhaustive search of the circuit to extract any feasible subcircuits. To reduce runtime, circuit partitioning, a limit on the size of feasible subgraphs, and other heuristic methods are proposed. A similar approach based on an optimised exhaustive search of the netlist is presented by Shi et al. [63].

However, these approaches are difficult to apply to hardware reverse engineering, as only small modules can be extracted, and exhaustive netlist searches are computationally expensive. This thesis closely follows and extends the functional approach presented by Li et al. [43]. Complementary to this thesis, which focuses on purely combinational circuits, Shi et al. [62] and Meade et al. [47] provide methods to reverse engineer sequential circuits through the extraction of Final State Machines (FSMs).

To detect the data path within the circuit, word-level structures are identified, by grouping both functionally or structurally similar gates [43, 65]. The concept of using cut enumeration as a heuristic to specify the functional properties of a cell in a combinational circuit is commonly used in the field of logic synthesis for technology mapping (also known as cell-library binding) [14, 19, 54]. The idea was first presented with regards to functional reverse engineering by Subramanyan et al. [67], who use cut enumeration to group functionally equivalent cells. An algorithm for efficient calculation of the set of k-feasible cuts is proposed by Chatterjee et al. [14], who provides a recursive, bottom-up approach. Further improvement in both runtime and scalability is suggested by Cong et al. [19], where the possibilities of cut ranking and pruning are discussed. Mishchenko et al. [49], likewise, propose an improvement to the existing algorithms through the use of signatures for cut filtering, which leads to a considerable decrease in both the runtime and memory use. In this project, a iterative approach, tailored to the use case, is employed.

In order to establish equivalence between cuts of nodes, boolean matching can be employed, which is commonly utilised in cell binding to match nodes to a cell library [40, 50]. Boolean Matching seeks to establish whether two boolean functions are equivalent, independent of Input/Output (I/O) permutation or negation. Two separate approaches exist depending on the application, either for library binding or for logic verification [40, 50, 61]. For the problem of cell library binding, where a set of single output functions with a small set of inputs ($k \leq 10$) is compared, commonly a conclusive, input permutation independent canonical form is calculated. A vast amount of literature regarding boolean matching for library binding can be found; the topic has been

present since the late 1980's due to its importance to technology mapping. Benini and De Micheli [8] offer a good overview of several proposed methods until 1997. Significant early contributions include [12, 37, 37, 61]. In 1999, the idea a calculating a permutation independent representative (P-representative), independent of functional properties, for permutation equivalence (P-equivalence) classes used in boolean matching was proposed by Debnath and Sasao [21]. A modified version of this paper in 2004 presents an updated view of this topic [22], including a computationally more efficient method for P-representative calculation. A similar approach to boolean matching is employed in this project, focussing on speedy calculation of a canonical form. Other solutions presented more recently are more specific to the required solution. For example, the method employed by Chatterjee et al. [14, 15] incorporates the calculation of equivalence classes directly into cut enumeration, which is carried out on a precomputed "Network with choices"; a process which is not scalable for larger number of input variables.

Circuit Clustering

To further support the candidate subcircuit enumeration process, circuit clustering can be used to locate possible functional modules within the netlist. Clustering algorithms and metrics vary in form, speed and scalability; a comprehensive overview of clustering methods and application can be found in [60]. Hierarchical clustering, that is, the clustering of nodes within a network which are structurally close, is a common used heuristic, but does not scale well for large networks. More complex clustering algorithms have been proposed, such as flow based circuit clustering using Markov chains [71], the Girvan-Newman algorithm, based on the removal of edges with the highest edge-betweenness [29], or the Louvain method which focussed on optimising the modularity of a graph [9].

While software reverse engineering commonly employs clustering to recreate understanding and allow maintenance of legacy software [2, 7, 39], in hardware systems, graph clustering was first introduced for circuit partitioning and Very-Large-Scale Integration (VLSI) system synthesis. It was though that modern VLSI systems would be too complex for layout synthesis algorithms and performance optimisation [18, 34]. To reduce the problem size, circuit clustering was used to allow for aggregation analysis of the netlist. Through grouping of clusters, the size of the network could be reduced, allowing for easier circuit partitioning [18]. An early approach, proposed in 1991 by Cong et al. [18], introduces the idea of random walks for circuit clustering. This idea was further refined in 1992 by Hagen and Kahng [34]. Circuit partitioning for performance optimisation during synthesis remains the main motivation for circuit clustering in modern literature, including a connectivity based approach using edge separability [17], and a fuzzy clustering approach [79]. In this thesis, three main approaches to circuit clustering are used: the Markov Clustering (MCL) algorithm [71], the Louvain method [9], and the Girvan-Newman algorithm [29].

Circuit Identification

Li et al. [43] presents the idea that the set of candidate subcircuits can be identified through Combinational Equivalence Checking (CEC) against a pre-defined cell library. The topic of CEC has been discussed comprehensively in literature, as it is an imperative part of circuit verification. A important requirement for CEC is a I/O correspondence; again, boolean matching can be employed. For logic verification, where the correspondence of inputs and outputs is unknown, and where commonly several outputs and a larger set of inputs per output exist, I/O equivalence can be established through the use of signatures [8, 22, 40]. Although Boolean Comparison can be inconclusive due to aliasing, it is a commonly used method when the computation of a canonical representation is too computationally expensive, or the function contains too many inputs [8]. It seeks to reduce the amount of feasible permutations, rather than establish a unique correspondence. An overview and comparison of common signatures is offered by Mohnke and Malik [50], who propose a Reduced Ordered Binary Decision Diagram (ROBDD) based method to identify signatures with minimum aliasing. The limits of signature based, permutation independent I/O Matching, in particular in regards to aliasing, are detailed by Mohnke et al. [51]. Concerning reverse engineering, the idea of a signature based approach for semantic matching for the identification of high-level functional components is proposed in 1998 by Doom et al. [24]. Other approaches to I/O matching for functional identification are presented in [41, 42], where Linear Temporal Logic (LTL) is used to calculate a unique pattern for each output during simulation using execution traces.

Once a I/O correspondence has been established, this thesis implements three different approaches to functional matching: Satisfiability (SAT) solving, Quantified Boolean Formula (QBF) solving and Satisfiability module Theories (SMT) based solving. Both SAT solving, and, more recently, QBF solving have become standard in modern hardware verification and CEC [30, 31]. A vast amount of literature exists for implementations of SAT and QBF Solvers, a good overview can be found in [32, 55, 73]. The idea to use templating combined with SMT solving for the reverse engineering of synthesised netlists is first presented by Gascon et al. [28], but both ideas originated much earlier. SMT based CEC has become more common in the field of hardware verification since the establishment of the SMT-LIB version 2 [48, 74], and SMT based templating was proposed for software generation in 2013 [1].

Contribution

This thesis seeks to refine and extend existing concepts presented in previous papers on reverse engineering of synthesised netlists ([28, 43, 65]). A heavy focus is placed on implementation and feasibility. In particular, the following contributions are made:

- *A refined approach to word and data path identification,* with less focus placed on control wires introduced through circuit optimisation. In past research, the criteria for feasible word identification is often based on common control wires. Here, a different set of criteria is used, to reduce the importance of control wires created through optimisation at synthesis.

- *An extension to previous subcircuit enumeration concepts through cluster based circuit partitioning.* In particular in highly interconnected netlist, the complexity of extracting a clear data path, and thus feasible candidate subcircuits, can be reduced by introducing additional criteria, in the form of circuit clusters.
- *A signature based approach to I/O matching for SAT and QBF solving.* While signature based I/O matching is commonly used in hardware verification, it has not yet been tested for reverse engineering processes. In particular the size of the component library, and thus the number of components which must be matched for each candidate subcircuit provides new complexity.
- *A comparison of three separate methods for subcircuit identification,* including SAT solving, QBF solving, and SMT based solving, using templates.
- *An exclusively open source implementation.* Many of the past projects have been funded by military and defence focussed organisations, are thus commonly concentrated heavily on theory. Due to the sensitive nature of the topic, detailed information regarding implementation is commonly not published.

Chapter 4: Methodology

In the following sections, an overview regarding the requirements, theories, implementations and computational complexity of the steps required to reverse engineer a netlist is given. One of the main difficulties of the concepts presented is the trade off between the generation the maximum amount of information, versus the complexity of refining and extracting the correct information. In this thesis, the approach taken is that information should be generated as fully as possible, even if this creates additional runtime complexity to refine. Especially as the netlist is unknown, and thus the user has no information of where possible words, clusters, or subcircuits may lie, this information should not be disregarded unless absolutely necessary.

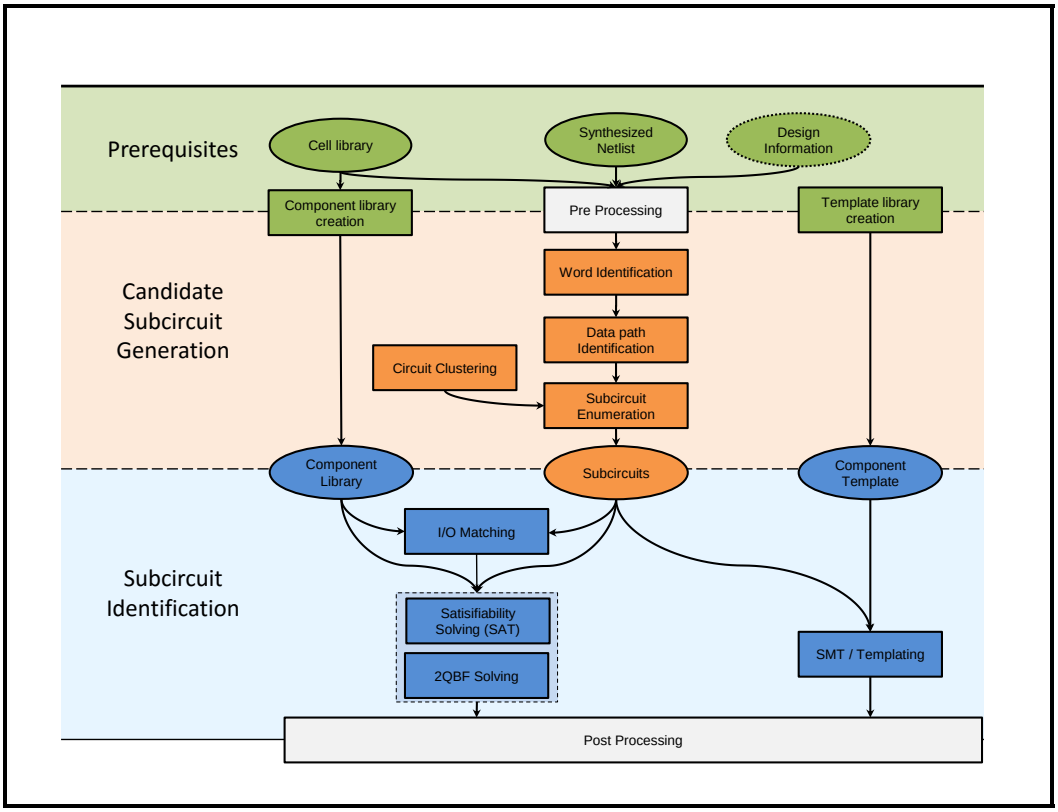


Figure 4.1: Design Flow

Figure 4.1 illustrates the relation between the algorithms of the two main phases: candidate subcircuit generation and subcircuit identification. Additionally, the three

main prerequisites for this project are shown. Primarily, the synthesised netlist of the acyclic, combinational circuit design under observation must be available. Additionally, the cell library used for the synthesis process, or if unknown, a cell library of similar functionality, is required for the parsing of the netlist and the generation of the component library. Finally, should further design information, such as groupings of primary inputs, groupings of primary outputs or manually identified control wires exists, these should be recorded in a design file exclusive to the given design.

4.1 Candidate Subcircuit Enumeration

4.1.1 Word Identification

Although the synthesis of a behavioural design removes high-level structure, and often the synthesis output is random, certain regularities remain [43]. In order to re-create structure in a synthesised netlist, the functional and structural aspects of each node in the design can be analysed. Nodes possessing equivalent structural or functional characteristics may be grouped into word level structures; it can be inferred that they may represent part of a word-level operation. It must be noted that gate sharing during synthesis can complicate the identification of words, consequently gate sharing is disabled during synthesis. Functional sharing through design choice is not affected.

4.1.1.1 Structural Characteristics

Often, nodes which are a part of an operation originally described at word-level pre-synthesis, will exhibit structural similarity post-synthesis. This is true in particular if all nodes perform an operation bitwise. For example, a boolean AND operation on 8 bits will often be broken into eight single AND nodes, generally, of the same driver strength. More complex operations display similar behaviour. To reconstruct these word-level operations, it is possible to analyse the structural characteristics of a node by inspecting the predecessors of the node.

An efficient metric for structural similarity is proposed by Li et al. [43] with the "shape" of a node. This idea is applied in this thesis, where firstly the "structure" of a node is defined as the directed graph of all nodes in the transitive fan-in of the nodes (i.e. all predecessors). The shape is then defined as the list of visited nodes in a reverse depth-first search (DFS) of the structure of a node, up to a predefined depth k. The size of k is restricted to $k \in \{2, 3, 4\}$; deeper searches often cause structural dissimilarities in otherwise similar nodes, as the search may traverse module or operation boundaries [43].

As nodes are grouped based on their shape, it is important that isomorphic structures are described in the same way. To ensure that isomorphic structures can be grouped, immediate predecessors of a node should be consistently ordered in a identical fashion during the DFS. Predecessing nodes are ordered alphabetically according to node type [43]. Since nodes of equivalent type will lead to an inconclusive ordering, these are further ordered according to the number of predecessing nodes (up to depth k), and as a final tiebreaker, by alphabetical ordering of immediate predecessors. Further criteria may be introduced to ensure complete graph isomorphism, but the above criteria proved to be adequate for this thesis.

Example 4.1.1. The concept of node structure and node shape is demonstrated in Figure 4.2, which shows the structure of node "NOR_4". A DFS begins at the node under test, "NOR_4"; this is the first visited node, and thus the first entry to the shape. As both predecessors of node "NOR_4" are of the same node type, the decision of which node should be pursued first is based on the number of predecessing nodes up to a depth k. In this case, node "AND_3" has six ancestors, while node "AND_4" has four ancestors. Consequently, the path past node "AND_3" is chosen first, and the visited nodes type is recorded in the shape. The search continues down the structure, splitting multiple ancestor nodes alphabetically. Once the search has exhausted the "AND_3" branch to depth k, it continues past node "AND_4", resulting in the shape seen on the right. Note that Primary Inputs (PIs) are also included in the shape.

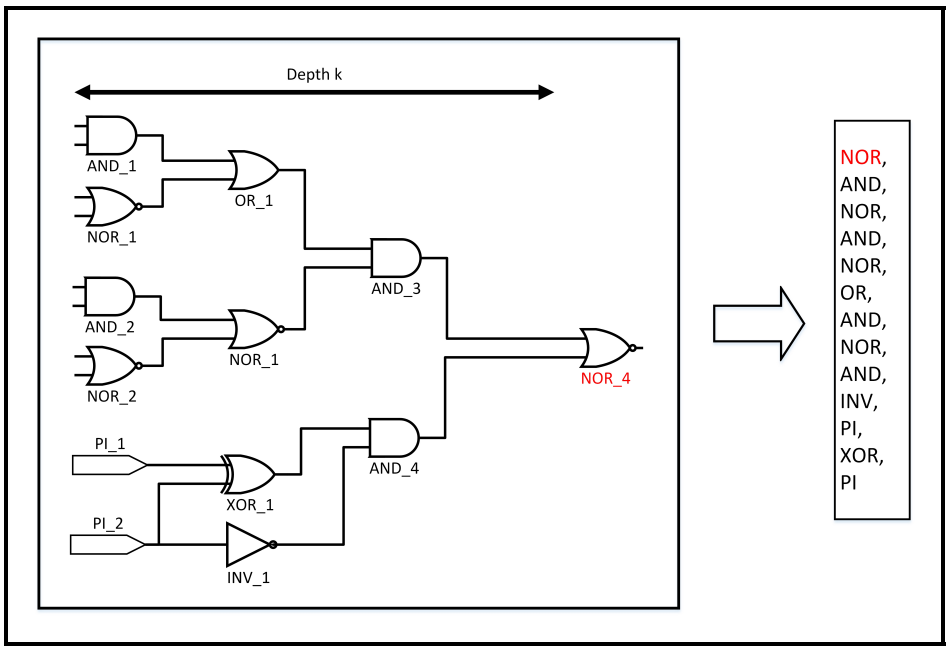


Figure 4.2: Structural Word Identification

Once each node has been assigned a shape, a hash function is created over all shapes to identify equivalent nodes. When a set of nodes is keyed to the same shape, the nodes are grouped into a candidate word. This method is a fast and direct approach to word identification, the computational complexity per node is defined by the Depth-First Search and is $\mathcal{O}(1)$. Although the complexity of a DFS is $\mathcal{O}(|V| + |E|)$, where $|V|$ is the number of nodes in the structure of node n, and $|E|$ the number of edges in the structure, this is limited by search depth k. For example, using 6-input nodes and $k = 4$,

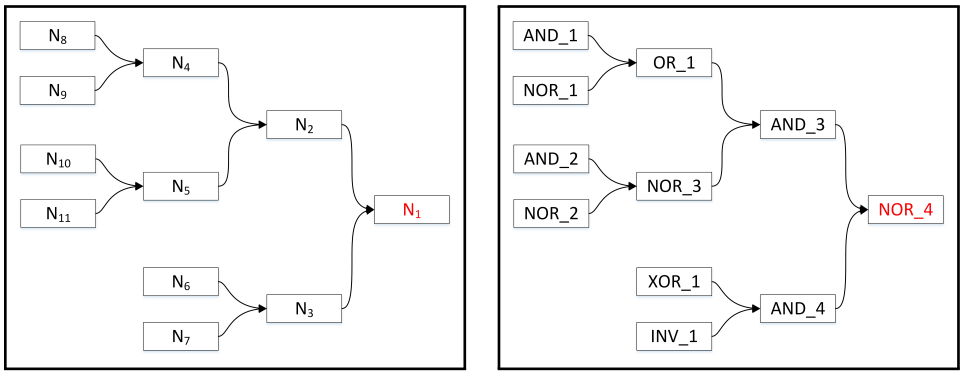
a maximum of 1,555 nodes and 1,554 edges can be transversed. The entire algorithm has a runtime complexity of $\mathcal{O}(n)$, running once for each node n.

This method is limited by its direct approach to node equivalence, as, especially with the use of larger cell libraries containing multi-function gates (such as And-Or-Inverter (AOI) cells or Or-And-Inverter (OAI) cells), structure may not be retained during synthesis. For example, a XNOR function operating on a set of bits may be synthesised mostly with XNOR type nodes, but for some bits, the functionality may be created using other node types. Using this approach, only those cells with equivalent node type representations can be grouped. Additionally, more complex functions may be implemented differently for individual bits, depending on the output of each bit, the required driver strength, and surrounding structure, while representing the same functionality.

4.1.1.2 Functional Characteristics

Using structural characteristics to group equivalent nodes into words requires structure to be completely retained during synthesis, severely limiting it's use in more complex circuits. Instead analysing the functional characteristics encounters no such difficulties; no matter which implementation of a function is chosen during synthesis, the functionality must remain the same. The field of technology mapping introduces the idea of cut enumeration to quantify the functional characteristics of a node [14, 15, 19]. A feasible cut is defined as a set of nodes in the transitive fan-in, such that any path from a primary input to the node passes through these nodes, and thus completely determines the function of the node [15]. The set of k-feasible cuts is defined as all cuts containing up to k inputs, starting with the redundant cut of size one, which contains the original node [14, 15]. These form a set of boolean functions with a single output, and k inputs, completely defining the functionality of the node. Although the size of the set of boolean functions for a number of nodes n is, at the maximum, $\mathcal{O}(n^k)$ [19], generally, far less feasible cuts are observed, and thus, conventionally, $4 \leq k \leq 6$ is used [15, 65].

Example 4.1.2. In Figure 4.3a, which illustrates the structural characteristics of node N_1, disregarding of any functionality, the set of 4-feasible cuts of node N_1 is $\{N_1\}$, $\{N_2, N_3\}$, $\{N_2, N_6, N_7\}$, $\{N_3, N_4, N_5\}$, $\{N_4, N_5, N_6, N_7\}$, $\{N_3, N_5, N_8, N_9\}$, $\{N_3, N_4, N_{10}, N_{11}\}$. Adding functionality to the structure, as in Figure 4.3b, allows for the computation of cut functions. The function of the previously calculated 4-feasible cut $\{N_2, N_3\}$ is $\neg(\text{AND\_3} \vee \text{AND\_4})$, the function of the cut $\{N_3, N_4, N_{10}, N_{11}\}$ is $\neg(\text{AND\_4} \vee (\text{OR\_1} \wedge \neg(\text{AND\_2} \vee \text{AND\_1})))$.



(a) Node Structure **(b)** Node Functionality

Figure 4.3: Node Structure for feasible cut calculation

Algorithm 1 Calculate set of k-feasible cuts

```
1:  procedure CALCULATE_K-FEASIBLE_CUTS(Nodes graph_nodes, constant k)
2:      hash-table result
3:      while len(result.keys) < len(graph_nodes) do
4:          unprocessed_nodes ← topological_sort(graph_nodes not in result)
5:          for each node n in unprocessed_nodes do
6:              if n is primary_output then
7:                  if predecessor of n in result then
8:                      result[n] ← result[predecessor of n]
9:              else if n is primary_input then
10:                 result[n] ← n
11:             else
12:                 func ← boolean_function(n)
13:                 if all inputs of func in result then
14:                     cut_set ← n
15:                     in_1 ← first input of func
16:                     in_2 ← second input of func
17:                     for each cut cut_1 in result[in_1] do
18:                         for each cut cut_2 in result[in_2] do
19:                             new_cut ← func.compose(in_1 : cut_1, in_2 : cut_2)
20:                             if degree(new_cut) <= k then
21:                                 cut_set.append(new_cut)
22:                         end for
23:                     end for
24:                     result[n] ← cut_set
25:         end for
26:     end while
27:     return result
28: end procedure
```

The set of k-feasible cuts can be calculated iteratively using algorithm 1. The hash table *result* is filled with $(key,\ value) = (node,\ cut\_set)$ pairs. The **while** loop (1.3) will run until *result* contains an entry for each node. Within the **for** loop, running over all unprocessed nodes, there exist three options for each node n:

- n is a Primary Output (PO), with a single input. The set of k-feasible cuts of n is defined as the set of k-feasible cuts of the predecessor of n (1.6).
- n is a primary input, with no inputs. The set of k-feasible cuts of n is the trivial cut n (1.9).
- n is a node representing a gate and function within the netlist, with a set of inputs. The set of k-feasible cuts of n is defined as the union of the sets of k-feasible cuts of the predecessors of n, as well as the trivial cut n. It is only defined if all inputs have a pre-calculated set of k-feasible cuts (1.11). If this is not the case, the algorithm will keep running, until all inputs are defined.

Example 4.1.3. The set of k-feasible cuts for node "AND_3" in Figure 4.3b can be calculated as follows using algorithm 1:

1. The boolean function of node "AND_3" is calculated: $And(OR\_1, NOR\_3)$ (1.12).
2. The k-feasible cuts of nodes "OR_1" and "NOR_3" must be contained within *result* (1.13). Otherwise, the algorithm will continue to loop until this is the case.
3. "AND_3" is added to the *cut_set* as the trivial cut (1.14).
4. The inputs or variables of the boolean function of node "AND_3" are initialised: $in\_1 = OR\_1$ and $in\_2 = OR\_3$ (1.15).
5. A set of **for** loops iterates over the set of k-feasible cuts for each input: e.g. $cut\_1 = Or(AND\_1, NOR\_1)$, $cut\_2 = NOR\_3$ (trivial cut) (1.17).
6. A *new_cut* is found for node "AND_3", by composing the boolean function:
   $new\_cut = (And(Or(AND\_1, NOR\_1), NOR\_3))$ (1.19).
7. The *new_cut* is tested for k-feasibility, the degree or number of inputs must be $\leq k$. If the *new_cut* is k-feasible, it is appended to the *cut_set* (1.20).
8. Steps 6 - 7 are repeated until every set of cuts has been composed.

9. The *cut_set* is written to *result* (1.24).

As nodes are processed in the **for** loop (1.5) only if all inputs to the node are defined, and thus only if the sets of k-feasible cuts for all inputs are defined, runtime can greatly increase with an unsuitable ordering of the nodes. For example, if the first unprocessed node is a primary output, the **for** loop (1.5) is unable to process this node until the predecessing node is processed, resulting in additional iterations of the algorithm, the computational complexity is $\mathcal{O}(n^2)$. Runtime is significantly decreased when unprocessed nodes are ordered topologically from the bottom up before each consecutive **for** loop (1.4). In fact, in most cases, with this new ordering of nodes, the **while** loop carries out only one iteration, reducing the time complexity to $\mathcal{O}(n)$

The illustrated algorithm is exemplary for two input boolean functions. Further input sizes can be achieved by stacking of **for** loops, based on the same cut composition theory. The number of input sizes is naturally restricted by the input size of cells contained within the cell library. Furthermore, many improvements to this algorithm exist. As discussed in chapter 3 , cut pruning and ranking [19], cut filtering [49], and a symbolic, Binary Decision Diagram (BDD) based approach can significantly impact runtime and memory usage. But these methods reduce the number of feasible cuts per cell, which may be useful for cell library binding, but is not advantageous when more than one feasible cut is used to find equivalence between cells. An entire set of k-feasible cuts per cell, rather than just a single cut, provides additional information regarding functionality and allows for a greater range of cell comparisons. Furthermore, the BddCut method focusses mainly on k-feasible cuts with $k>$ 6, and is thus not applicable to this problem, which focuses on feasible cuts of $k <= 6$.

4.1.1.3 Calculating a Permutation Independent Representative

To allow for comparison between nodes of the same functionality, but with differing input permutations, each feasible cut of each node is expressed using a permutation independent representative (P-representative). As discussed in chapter 3, a fast, efficient, but memory intensive method is proposed by Debnath and Sasao [22]. The method is practical for up to eight inputs, and thus suitable for the problem presented here [22]. The algorithm groups functions into P-equivalence classes, if they have identical P-representatives. Any cut within a given P-equivalence class can be transformed into another by rearranging of inputs, and thus the functionality is identical.

Each cut can be represented by its binary truth table output vector, for a given input encoding. Permuting the inputs will lead to a different output vector, which can be represented by moving the original truth table vector coefficients. To calculate the P-representative, first, all possible permutation coefficients of the truth table of a n-input function are precomputed. As can be seen in Table 4.1, a three input function has six possible permutations, with a corresponding set of eight truth table coefficients for each permutation. For the given input encoding, starting with c_o for inputs "000" and ending in c_7 for inputs "111", coefficients c_o and c_7 will remain in the same place, as a permutation of inputs cannot affect the output. All other output coefficients will move positions depending on the input permutation. The calculation of the truth table

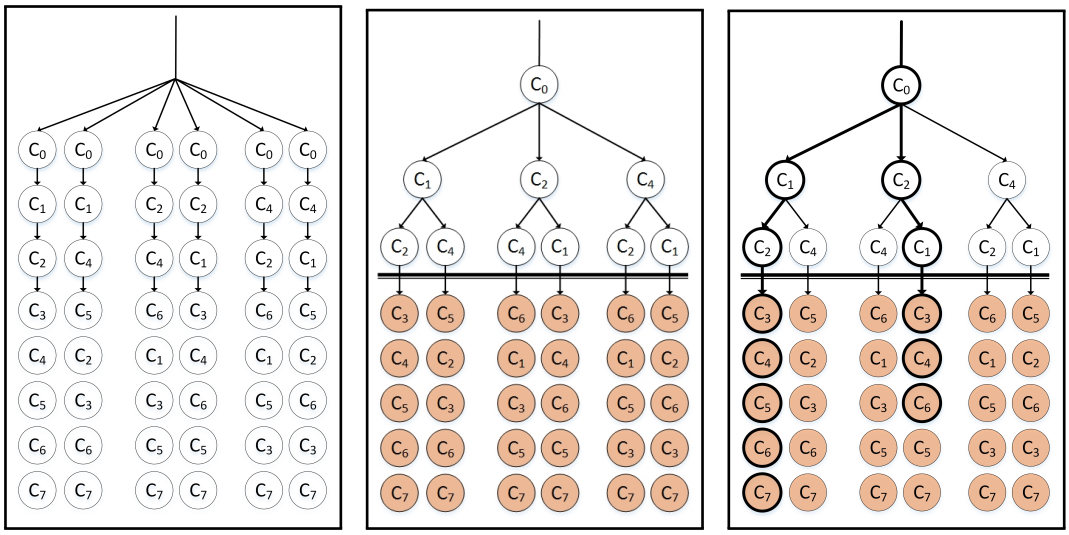
coefficients is resource intensive, because for each n-input function, $n!\ 2^n$ positions must be computed [22]. The function with the smallest binary truth table output vector of a P-equivalence class is used as the P-representative [22], i.e., the function with the smallest numerical representation. This can be translated to mean that all "1"s within the output truth table vector are moved as far as possible towards the least significant bit (LSB) when permuting inputs.

$f(x_1, x_2, x_3)$	$f(x_1, x_3, x_2)$	$f(x_2, x_1, x_3)$	$f(x_2, x_3, x_1)$	$f(x_3, x_1, x_2)$	$f(x_3, x_2, x_1)$
c_0	c_0	c_0	c_0	c_0	c_0
c_1	c_2	c_1	c_4	c_2	c_4
c_2	c_1	c_4	c_1	c_4	c_2
c_3	c_3	c_5	c_5	c_6	c_6
c_4	c_4	c_2	c_2	c_1	c_1
c_5	c_6	c_3	c_6	c_4	c_5
c_6	c_5	c_6	c_3	c_5	c_3
c_7	c_7	c_7	c_7	c_7	c_7

Table 4.1: Precomputed Coefficients for three Input Function

This table can be transformed into a tree of all precomputed coefficients, as illustrated in Figure 4.4a. With a breadth-first search (BFS) search on this tree, permutations which cannot lead to the P-representative can be detected early and discarded. As both the set of possible permutations (and thus the truth table coefficients), and the BFS trees can be precomputed, neither has a direct effect on overall runtime. Consequently, only the BFS itself contributes to the computation complexity for calculating a P-representative, which usually requires $\mathcal{O}(|V| + |E|)$ calculations. This can be reduced with the partitioning of the search tree, as illustrated in Figure 4.4b. For example, in the case of a three input function, instead of of six possible branches, only three must be searched to conclude a definite result [22]. Subsequently, the number of vertices $|E|$ is reduced to $|V| - 1$, which is dependent on the number of inputs to the feasible cut k, giving a complexity of $\mathcal{O}(k)$. As the P-representative of each feasible cut of every node is calculated, the computational complexity of the entire process is $\mathcal{O}(n^{2k}k)$.

Example 4.1.4. For the function $((a \vee \neg b) \wedge \neg c)$, with the truth table output vector 10001010, coefficients c_0, c_4 and c_6 are "1", all others are "0". In order to find the smallest binary representation, a BFS starts at coefficient c_0, as illustrated in Figure 4.4c. This means that all possible representations start with a 1*. Of the succeeding nodes, both c_1 and c_2 lead to a representation starting with 10* while choosing c_4 will lead to a representation starting with 11*. Thus, coefficient c_4 cannot lead to the smallest representation. In the next layer of the BFS, coefficient c_4 is again eliminated. This process continues, until coefficient c_6 is reached on the right hand path, eliminating this path (which would lead to a p-representative of 10001100). Consequently, the p-representative of the function $(a \vee \neg b) \wedge \neg c)$ is 10001010.



(a) BFS Graph **(b)** Partitioned BFS Graph **(c)** Example Search

Figure 4.4: Permutation Coefficents for three Inputs Function

Once a P-representative has been calculated for each cut, a hash table over all P-representatives can be created, and when more than one node is keyed to the same P-representative, this set of nodes is grouped into a candidate word. 2-feasible cuts, including the trivial cut, are not included during grouping. A feasible cut with two inputs represents the function type of the node, and will cause all nodes with an equivalent node function to be grouped into a candidate word.

A special case of candidate word occurs when a node within a P-equivalence class is an input to the P-representative of another node within the same class. If this repeats within the P-equivalence class, it can be deduced that the set of nodes constitutes a carry chain. This concept is illustrated in Figure 4.5. Consider nodes "OR_0" and "OR_1", with the respective feasible cuts "((XOR_1 and (NAND_1 nor OR_1)) or OR_1)" and "((XOR_2 and (NAND_2 nor OR_2)) or OR_2)". Both nodes will be grouped into a P-equivalence class, and each node is an input to the feasible cut of another node within the class. If this is true for a large subset of the nodes within the class, the class may constitute a carry chain, where information is passed along nodes "OR_0", "OR_1", "OR_2" and so on.

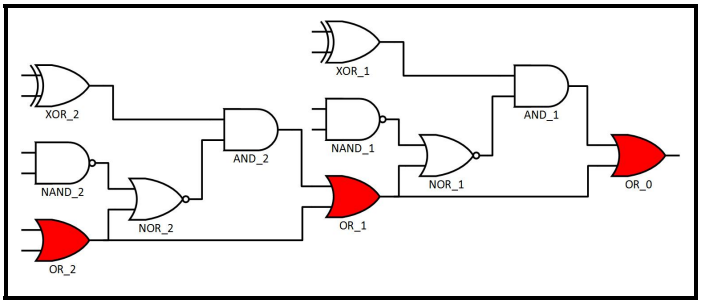


Figure 4.5: Identification of Chain Words

Furthermore, once a set of nodes has been grouped into a P-equivalence class based on the functional equality of each nodes respective feasible cut, further words can be identified by comparing which permutation of inputs leads to the P-representative. For example, consider the bitslices of functionally equivalent nodes "OR_X1" and "OR_X2" in Figure 4.6. The permutation of inputs which leads to the P-representative is the same for both functions, ie, "OR_1" : "OR_2", "NAND_1" : "NAND_2", "XOR_1": "XOR_2", and the cell type of corresponding input nodes is also equivalent. Subsequently, corresponding nodes can be grouped into new candidate words, as they are inputs to functionally equivlant bitslices.

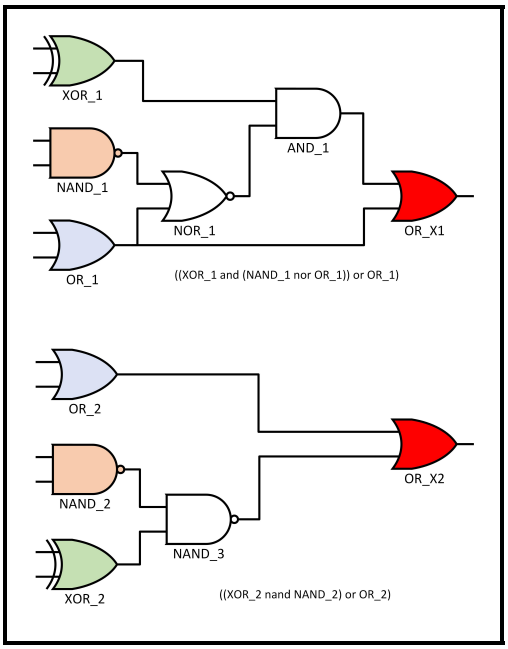


Figure 4.6: Identification of Input Words

This method is far more exhaustive than structural word identification, taking into account functional rather than structural characteristics, which are not removed during synthesis. However the runtime is significantly larger than when using a more basic method. Several algorithmic steps are required, first for the identification of a set of k-feasible cuts for each node, followed by the calculation of a P-representative for each cut, and finally a grouping based on a hash-table. In the worst case, the computational complexity is $\mathcal{O}(n^{4k}k)$.

4.1.1.4 Identifying feasible words from candidate words

A feasible or true word is defined as a set of nodes working as a unit to carry out a specific function concurrently, in the sense that data will propagate along a path of consecutive words within the circuit. The above methods identify a multitude of words, of which some may not entirely fit this definition. As discussed, this thesis seeks to retain the maximum amount of information, yet, at this point, the set of words must be pruned to extract only true words, and discard all others. In previous projects, the

criteria was often based on common control wires [43], here another set of criteria is used. The above definition allows for the specification of a set of criteria for feasible words as follows:

- The nodes within candidate words are examined for *hierarchical independence*. No node of a word should exists in the transitive fan-in or fan-out of another node of the same word, with the exception of chain words, where this characteristic is explicitly desired.
- Each candidate word is inspected with regard to the *structural distance* between the nodes, as defined in section 2.1. It cannot be assumed that a given function or structure is contained only in one word within a given netlist. Often, both functionality and structure may be repeated within the circuit. To ensure that the nodes in a word are within the same general area of the circuit, the distance between each node in each word should not exceed a given constant relative to the size of the word and the graph [43].
- The number of nodes contained within the candidate words should lie within a predefined set of *allowed sizes*, to avoid unnecessary calculation steps through the discovery of unrealistic or incorrect words. A common lower boundary is 4 bits; for larger circuits, this boundary may be higher. The chosen upper bound strongly depends on the number of Primary Input (PI) bits and the size of the netlist in question. For example, it is highly unlikely that a circuit with 16 PI bits and 8 Primary Output (PO) bits will contain a word with 32 bits.

These criteria are applied to all candidate words; words which satisfy these conditions in full are defined as feasible words. All other words are re-examined to test whether a subset of the word fulfils the criteria. If such subsets exist, a resolution function is used to find the largest coverage of subsets over the full candidate word, the resulting subsets are also declared as feasible words. This allows for identification of separate words which carry out the same function several times across the circuit, but are not related. These words are grouped into a single word within the candidate word identification process, the checking algorithm resolves this larger single candidate word into a set of smaller, feasible words.

Finally, often the above algorithms will generate a large set of feasible words, which may cover some nodes in the netlist more than once, or may even fully contain other words. This means, that two or more words may contain the same node(s), which should not occur within a synthesised netlist without gate sharing. To reduce this overlap, first, any word which is a full subset of another word is identified and removed. Then, if possible, an exact cover of all nodes is found, otherwise a greedy algorithm can be used to find the best cover where each node is contained only once. Words in the solution to this problem are source words, and can now be used to identify a data path within the netlist.

4.1.2 Data Path Identification

Although some structural information of the synthesised netlist is gained through the extraction of a set of source word structures, no information of the relation between

these words is given. The next logical step in recreating structure is to create a hierarchy of words within the netlist, in order to determine how data travels through the circuit.

A simple solution is to order words according to their position within the netlist, as, in a acyclic, combinational circuit, information flows from the primary inputs to the primary outputs. If a set of paths exists between all nodes in word w_1 and all nodes word w_2, it can be inferred that data may travel between w_1 and w_2. Ergo, word w_1, which lies completely within the transitive fan-in of word w_2, can be ordered higher in the hierarchy of words, that is, a path exists from $w_1 \rightarrow w_2$ and (u, v) is an edge in the directed acylic graph (DAG) of the data path. But, whilst a possible word-level data path can be created by analysing the set of feasible words with regards to their structural position within the netlist and their interconnections, there is no guarantee that data will truly travel along this path. Although there may exists a structural connection between the nodes of two words within a graph, data may not travel between these words, due to the input assignments or control wire assignments. Furthermore, using this method, it becomes difficult to judge how far words are apart, and if other words lie on the path between the two words.

Subramanyan et al. [67] and Li et al. [43] propose a method to ensure that a path between two words is only recorded if data can be propagated from a word to the next, removing the possibility of incorrectly recording paths between two words which are only structurally connected. The algorithm used in this thesis closely follows this method, using symbolic simulation [11] and Roths D-algorithm [57] to propagate values from a given word to the next. Although this method is computationally far more expensive than the less complex structural analysis, it results in a definite data path, and allows for the discovery of new words, greatly outweighing the computational disadvantage.

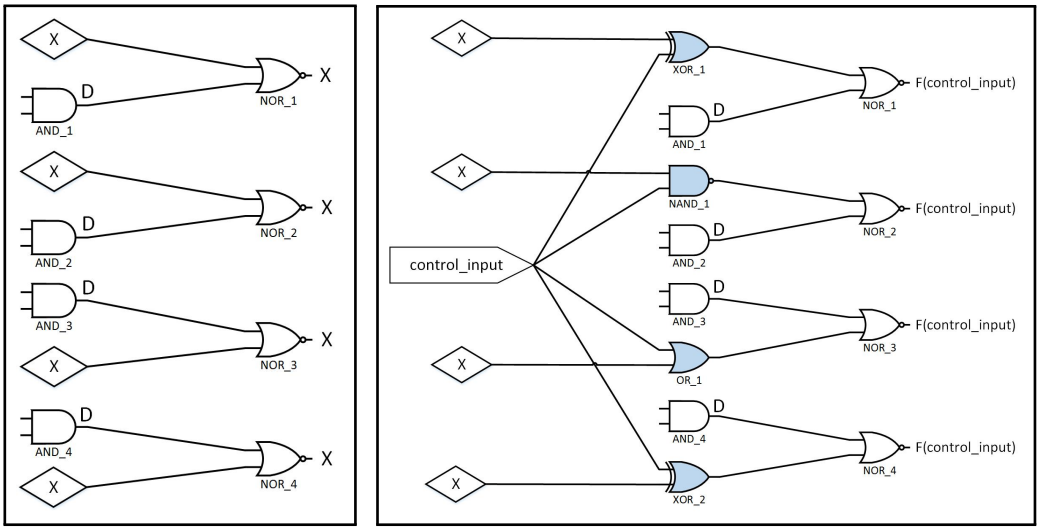
The method attempts to propagates each source word exactly one step forward or backward in the design, creating a new candidate word consisting of nodes of identical function. All feasible words are queued, for each set of new candidate words exactly one step forward or one step backwards in the design is found. A candidate word consists of nodes of identical functionality, and should be of no greater cardinality as that of the original word. In the case of forward propagation, if arbitrary values applied to the source word can reach this candidate word, it can be assumed that data can propagate from one word to the next, and a path from source to candidate word is recorded. To test this assumption, symbolic evaluation, similar to to Roth's D-calculus is used [11, 43, 57]. This extends the functionality of each cell type to include the values D and $\overline{D}$, representing a symbolic value, which must not be masked for the word to be propagated. The new functionality of a XOR cell is demonstrated in Table 4.2.

a	b	$a \oplus b$	a	b	$a \oplus b$	a	b	$a \oplus b$	a	b	$a \oplus b$
0	0	0	0	D	D	0	$\overline{D}$	$\overline{D}$	$\overline{D}$	D	1
0	1	1	D	0	D	$\overline{D}$	0	$\overline{D}$	D	$\overline{D}$	1
1	0	1	1	D	$\overline{D}$	1	$\overline{D}$	D	D	D	0
1	1	0	D	1	$\overline{D}$	$\overline{D}$	1	D	$\overline{D}$	$\overline{D}$	0

Table 4.2: Re-defintion of XOR cell for symbolic evaluation

To evaluate whether the symbolic value D can be propagated from the source

word to the new candidate word, or vice versa for backward words, it is not sufficient to assign the source word a symbolic value. The output would be undefined for all nodes with more than one input, as all other inputs of the candidate word are assigned an undefined value X, as is illustrated in Figure 4.7a. Here, the original word is ("AND_1", "AND_2","AND_3","AND_4"). Inputs to the forward word ("NOR_1", "NOR_2","NOR_3","NOR_4") which are not connected to the source word, are assigned an unknown input "X", and thus "X" is propagated to the forward word.



(a) Word Propagation **(b)** Word Propagation with Control Inputs

Figure 4.7: Symbolic Evaluation for Word Propagation

Instead, it is assumed that there exist some set of control wires for the candidate word, that is, wires which are in the transitive fan-in of every node of the word up to a certain search depth k. These are are not part of the data path, but rather act as a control signal, for example the select signal of a multiplexer. Control Inputs from which a path leads to all nodes in the candidate words are included in this set of control wires. This is illustrated in Figure 4.7b, where the nodes in blue contain a path to a control input, and are thus included in the symbolic evaluation. If a control wire is identified, it is possible that for some concrete assignment of $\{0, 1\}$ to the control wire, the symbolic value can be propagated to the candidate word. Thus, once a possible word pair of source word and candidate word is determined, and control wires have been found, a netlist slice can be set up to contain the candidate word and all cells in the transitive fan-in which lie in the path to a control word.

Once all unknown signals are assigned as undefined X, symbolic evaluation can take place: all permutations of inputs for the control wires can be tested to detect an assignment of control wire inputs which will allow the output of the nodes in the candidate word to be either D or $\overline{D}$. If this succeeds, the candidate word is added to the queue of identified words, and undergoes the same operations to further propagated it through the design. The resulting relations between words can be recorded as a graph of words.

One of the main difficulties presented by this method is that, if no control wires are identified, and no control inputs exist for the netlist, propagation cannot take place except for single input nodes. While Li et al. [43] assumes that the synthesis tool will create additional control wires in order to optimise the netlist, this is often not the case in the circuits tested in this thesis. This means that particularly in circuits with no control wires, this method does not provide a valid representation of the data path. Furthermore, in many cryptographic modules, which focus on the scrambling of data, a clear data path simply cannot be identified. In these cases, the relation between words must be established using only structural heuristics.

Another problem is an accurate runtime estimation, as it cannot be foreseen how many new words will be generated and added to the queue. In particular, the search for forward and backward words has a significant impact on the runtime if number of inputs and outputs of the nodes in the source word is very high. For example, for a 16-bit word, where each node has four outputs of identical node type, $4^{16} \approx 4bn$ forward words are possible; this constitutes the worst-case scenario. Furthermore, for each forward and backward word, the criteria for feasible words must apply as defined in subsubsection 4.1.1.4. Especially removing overlapping words, and words which are complete subsets of other words can strongly influence the runtime.

However, establishing a valid datapath is vital for subsequent subcircuit enumeration, and thus, the computation complexity, while not ideal, can be condoned.

4.1.3 Candidate Subcircuit Generation

The creation of a data path greatly increases the understanding of the unknown netlist, but through the enumeration of possible subcircuits, and subsequent identification, information regarding the functionality of the netlist can be gained. As discussed in chapter 3, a multitude of methods to extract and enumerate candidate sub modules have been proposed. Many of these depend on exhaustive searches of the netlist. Often the identified sub modules contain only a limited number of nodes ($n < 30$). These methods are unsuitable for the reverse engineering of larger netlist, as exhaustive searches are computationally expensive, and often circuits contain a large number of nodes.

Instead, a more functionality based approach generates candidate subcircuits using previously identified words as module boundaries [43]. As combinational circuits often contain functions on word-level basis, it can be assumed that the cells between a set of words may form a functional module [43]. First, a set of input words and a set of output words is chosen from the data path, so that the input words are ancestors of the output words. Cells in the intersection of all descendants of the input words and all ancestors of the output words are carved out of the circuit, to create a potential subcircuit. Although in the case of many functional operations, the input and output words are of the same size, this is not always required.

It is difficult to extract functional modules, which will exactly match functional modules in a predefined component library, by using words as boundaries. Not only are modules implemented differently using different synthesis tools, different cells libraries or different high level description, but during synthesis, the circuit will often be optimised. This means that boundaries are often vague, or dependent on control wire inputs. For example: a arithmetic logic unit (ALU) may be be optimised to share gates, so that

the required combination of gates for a function depends on the correct operation code. Furthermore, due to the approach of employing word level-structures as boundaries, single bit inputs or outputs are not identified, and thus not included in the candidate submodule. In the case of a 4 bit Ripple-Carry Adder (RC Adder) with a single input carry bit and a carry output bit, only the cells between the two input words and the single output word are included in the subcircuit, whilst cells from the single input and leading to the single output will not be carved from the netlist.

Missing inputs or missing control wires will present in incomplete netlist slices, that is, nodes within the carved out module will not posses the required amount of inputs. For example, a two input cell may only posses one of the required inputs within the new submodule, because an input to the submodule has not yet been identified. Hence, once all combinations of input words and output words are compiled, not only are cells between these words identified, but the resulting submodule is analysed and expanded to include any viable control wires or further inputs, until a complete combinational netlist slice is created. If the submodule cannot be expanded until all inputs are defined, it is disregarded. Remaining subcircuits are declared as feasible candidate subcircuits.

If only one step of cells lies between the input and output words, as may be the case if the output words were directly propagated from the input words, the resulting circuit is trivial. Words with greater structural distance between them, will contribute more information than smaller candidate submodules. On the other hand, for a larger subcircuit, there is also a higher chance that the subcircuit is incomplete, and will thus require more verification.

In line with the general tone of this thesis, once again the trade-off between information and correctness has a considerable impact on the result and runtime. As the size of the data path directly impacts the number of candidate functional modules, the importance of correctly identifying and propagating feasible words is again highlighted. When a single word, with a ancestors in the data path, acts as the output to a functional module, which has a maximum of c input words, $\sum_{i=1}^{c} \frac{a!}{i!(a-i)!}$ combinations of input words are possible, resulting in the same number of candidate subcircuits. The addition of a single incorrectly identified word in the data path will not only significantly increase the number of functional modules which must be generated, verified and matched, but in most cases, submodules containing this word will not be identified during the component matching step. On the other hand, any module boundaries which are not identified as word level structures will reduce the chance that the functional module is detected and identified.

For each verified candidate subcircuit, the functionality of each output is calculated to prepare for later component matching. Similarly to the feasible cut calculation in Figure 4.2, the function calculation is both computationally expensive and memory intensive. The runtime complexity is at the worst case $\mathcal{O}(n^2)$, where n is the number of nodes within the candidate subcircuit, as two loops over all nodes are required. With suitable ordering of the nodes, this can be reduced to $\mathcal{O}(n)$.

Circuit clustering is incredibly helpful in the candidate subcircuit verification and expansion process. Using only the data path for subcircuit generation means that each possible combination of input and outputs words must be generated, verified, expanded, and matched against the component library. As will be discussed in section 4.2 , pre-calculated circuit clusters may give an idea as to where potential functional subcircuits

are situated. Thus, words which lie within the boundaries of these pre-defined clusters can be prioritised, significantly reducing the time taken to identify and verify unsuitable candidate subcircuits. Furthermore, clusters can aid with the circuit expansion process, identifying possible inputs or outputs in the cluster boundaries, which are not covered by word level structures.

4.2 Circuit Clustering

Reverse engineering synthesised netlists is based on the idea that functional modules within the netlist can be detected and identified. In the previous sections, a functional method to reconstruct the data flow, and identify possible functional modules within the data flow is outlined. Another method which seeks to identify potential functional modules is circuit clustering.

Definition 5. *A **cluster** is a natural grouping within a graph.*

Clustering algorithms vary in form, speed and scalability, and, depending on the required outcome, different parameters can be chosen to create clusters. Hierarchical clustering, that is, the clustering of nodes within a network which are structurally close, is a commonly used method, but does not scale well for large networks. As netlists often contain many thousand nodes, only methods which scale well with large network sizes are viable. More complex clustering algorithms determine clusters based on density, modularity, distribution, or cliques within the graph. In the case of circuit clustering, nodes within functional groups will commonly be more connected than others.

Definition 6. *The **Modularity** of a graph measures the connectedness of node within a cluster compares to the connectedness of nodes between clusters [9].*

A high modularity signifies that nodes within clusters are highly interconnected, while connections between nodes within different clusters are sparse. In synthesised circuits, nodes of a functional module are not only located structurally near each other, but are often also more closely interconnected, allowing identification through clustering methods focused on modularity and interconnectedness.

One of the most promising algorithms is the Markov Clustering (MCL) algorithm, discovered by van Dongen [71]. It is especially suited for sparse graphs, and thus ideal for netlist analysis. The algorithm is based on the idea that a random walk within a graph will more often pass through related node than through unrelated nodes [72]. This means that a random walker has a higher probability to stay contained within a cluster of nodes than to leave [72]. This can be represented mathematically as a Markov chain, where the Markov matrix is the sparse, column normalised adjacency matrix of the graph, which shows the probability of the walker moving from one node to the next [45]. Each step of the random walk corresponds to one step in the Markov chain. In particular during the first steps of the walker, the probability matrix shows a strong clustering of nodes, during subsequent steps the flow becomes more distributed [45]. The MCL algorithm seeks to actively boost this phenomenon by increasing the number of neighbours of each node by stepping along the Markov chain, known as expansion, and to both increase the importance of high probability neighbours, and decreases that

of low probability neighbours through inflation [72]. A high value for inflation will lead to more finely grained clusters, while a high expansion value will generally increase the size of clusters [71].

These two techniques, expansion and inflation, are illustrated in algorithm 2. The amount of expansion and inflation are controlled by the constants e and r (2.1). While no steady state is reached (2.5), the matrix is expanded by stepping along the Markov chain e times (by finding $\mathbf{N} \cdot \mathbf{N}$), and then inflated, by taking the r power column-wise (2.12) [45, 72]. Once further multiplication yields no futher changes in the matrix, the probability matrix has reached a stable state [45] (2.5). Clusters can be formed by aggregating all nodes which contain an entry within the same row.

Algorithm 2 Markov Cluster Algorithm

```
1:  procedure MCL_CLUSTERS(matrix A, constant e, constant r)
2:      matrix A_SL ← A + I                          ▷ Adjacency Matrix with self loops
3:      matrix N ← normalise(A_SL)
4:      matrix C ← J          ▷ Matrix C records change to N, initially "1" matrix
5:      while C is not 0 do
6:          matrix N_old ← N
7:          constant e_const ← e
8:          while e_const > 1 do                                          ▷ Expand
9:              matrix N ← N · N
10:             constant e_const ← - e_const
11:         end while
12:         matrix N ← N^r                                                ▷ Inflate
13:         matrix N ← normalise(N)
14:         matrix C ← N_old − N
15:     end while
16: end procedure
```

A defining feature of this algorithm is that, while the number of clusters is not pre-defined, it can be adjusted by modifying the expansion and inflation factors. This means that the user does not need to pre-define how many clusters will be within the graph, but rather can adjust the granularity of the clusters by adjusting the constants accordingly. The MCL has a computational complexity of $\mathcal{O}(n^3)$, where n is the number of nodes in the graph. This can be further improved by taking advantage of the fact that the matrix is sparse, and through further pruning, until the complexity is $\mathcal{O}(nk^2)$, where k is the pruning factor [71]. MCL-Edge is a tool based on an optimised version of this algorithm [70].

To validate the identified clusters, and to allow for comparison, a second algorithm - the Louvain Method, originally described by Blondel et al. [9], is used. It focuses on speedy partitioning of networks to find the highest modularity within each cluster. It is a iterative, two step algorithm, which gives a good approximation of the optimal clusters [9]. In the preparation step, each node is assign its own community. Then, each community attempts to merge with a neighbouring community, based on which merge gives the highest modularity. Second, each community is re-evaluated with regards

to weight. This is repeated until no merges will lead to a higher modularity of the network, giving a maximum modularity [9]. According to Blondel 'exact computational complexity of the method is not known, the method seems to run in time $\mathcal{O}(n \log n)$' [10]. An existing python implementation is used [3].

Finally, a third approach, the popular Girvan-Newman algorithm is used [29]. It focussed on the boundary of clusters, rather than node within clusters. It is based on the idea that nodes with a high edge-betweenness, that is, nodes which lie between many pairs of nodes, can be iteratively removed, leaving behind clusters. Similarly to the MCL algorithm, this algorithm runs with a computational complexity of $\mathcal{O}(n^3)$ on sparse graphs. The algorithm produces a different, more modular set of clusters with each step, which is particularly helpful for the problem posed in this thesis, as the size and number of functional modules within the netlist is unknown. Again, an existing python implementation is used [38].

Unfortunately, it is nearly impossible for circuit clustering to detect full functional modules, which can be matched to a component library, and thus identified. Often, nodes may be included within the cluster which are not part of the functional module, or the module boundaries are not clearly defined. Moreover, while the assumption that the modularity of functional modules is higher in netlist is commonly correct, there are instances where this may not be the case.

So, while the circuit clustering algorithms described here cannot be used as stand-alone methods to detect and identify functional modules, they can be used to give support and validity to the other methods offered by this thesis. As previously mentioned, reverse engineering a netlist using the techniques described in this thesis is always a balancing act between too little information and potentially wrong information. For example, during word identification, words are retained or discarded based on their relative distance, but the distance cut-off is chosen manually. This puts the responsibility on the designer to choose a value which is neither too small to disallow potential words, or too large, causing unrelated nodes to be grouped into a word. Testing instead whether words are fully contained within a cluster can provide another criteria to check for the validity of words. Similarly, clustering methods can aid circuit enumeration, where words are chosen as potential boundaries to functional modules. Without circuit clustering, each possible combination of words is tested, each resulting subcircuit verified, and each verified subcircuit matched to the component library. Alternatively, identifying words within cluster boundaries may give an idea where possible subcircuits lie, significantly reducing the number of candidate subcircuits, while yielding a similar number of matches.

4.3 Subcircuit Identification

To identify the functionality of a candidate subcircuit, it can be compared and matched to a set of precompiled components. In the following sections, first the creation of the required component library is described. As the matching technique requires information regarding the I/O correspondence of the candidate subcircuit and component, a technique to identify viable I/O permutations is detailed in subsection 4.3.2. Finally, three methods to for Combinational Equivalence Checking (CEC) between a subcircuit and library component are discussed: Satisfiability (SAT) solving, Quantified Boolean Formula

(QBF) solving, and Satisfiability module Theories (SMT) based templating.

4.3.1 Component Library creation

To identify the functionality of the candidate subcircuits of a synthesised netlist, a comparison is made to the component in a predefined component library. In theory, the library should contain any conceivable combinational circuit, implementing any possible set of functionalities. In reality, the size of the component library is defined by the end-user, who must implement each included functionality. As noted by Li et al. [43], components representing general functionalities are of more use than highly specific circuits. The library used for this thesis was created with the test cases in mind. Currently, the following component types are included, with input bit sizes between 4 - 32 bits:

- Basic Boolean Functions
- Adders
- Subtractors
- Multipliers
- Dividers
- Multi- and Demultiplexors
- Cryptography cores
- ALUs
- Encoders and Decoders
- Incrementers and Decrementers

Each combinational circuit is fully specified, including the following information:

- Number of Inputs
- Number of Outputs
- if applicable, grouping of Inputs and Outputs
- Functional Verilog
- Synthesised Verilog
- Test Bench
- Formal Description of Input/Output Behaviour
- Graphical representation
- Signature Functions for each Output

Currently, 346 components are contained within the component library. Pre-computing both the boolean functionality and signatures for each output greatly decreases the overall runtime, as these calculation are computationally expensive. As with candidate subcircuits, the boolean function is calculated iteratively, with a runtime complexity of $O(n)$.

4.3.2 Input/Output Correspondence

To prove equivalence between a candidate subcircuit and a library component, knowledge regarding the correspondence of inputs and outputs is required [50]. Consider candidate subcircuit $c_k \in \mathcal{C}$ with Inputs $\mathcal{X}$, Outputs $\mathcal{Y}$, and functionality $\mathcal{Z} = z_1(\mathcal{X}), z_2(\mathcal{X}), \ldots, z_{|\mathcal{Y}|}(\mathcal{X})$, and an abstract component $\alpha \in \mathcal{L}$ with Inputs I, Outputs O, and functionality $\mathcal{S} = s_1(I), s_2(I), \ldots, s_{|O|}(I)$. An immediate disqualifying criteria for equivalence is a difference in the amount of outputs and inputs. Disregarding possible control wire side inputs, two circuits cannot be functionally equivalent unless the respective number of input bits and output bits are equivalent. Given that $|\mathcal{X}| = |I|$ and $|\mathcal{Y}| = |O|$, a set of input permutations $\mathcal{P}_i = (\pi_1, \pi_2, \ldots)$ and output permutations $\mathcal{P}_o = (\phi_1, \phi_2, \ldots)$ must be found, so that:

$$\exists \pi \in \mathcal{P}_i,\ \exists \phi \in \mathcal{P}_o,\ \forall z_i \in \mathcal{Z},\ \forall s_i \in \mathcal{S}\ :\ s_{\phi(i)}(\mathcal{I}) = z_i(\pi(X)) \tag{1}$$

A brute force approach, where equivalence is tested for every possible permutation of inputs and outputs, is not efficient. For a circuit with i inputs and o outputs, $i! \times o!$ permutations are possible; a 4-bit Adder with eight inputs and four outputs lead to almost a million possibilities [24]. Consequently, for larger circuits, using a brute force approach is not viable.

A common technique to establish the correspondence of output signals of two circuits is the use of a signature to compute boolean comparison. A signature of a boolean function describes a unique characteristic of the function, independent of input permutation or negation of inputs [8, 50]. Signatures are not necessarily unique to a specific function, a phenomenon known as aliasing [50]. Accordingly, the correspondence of a set of outputs may not be completely conclusive, but as a different signature for two functions precludes boolean equivalence, it can significantly reduce the search space. In fact, if two circuits do not have the same of signature functions for the outputs, no further matching is required, as the circuits cannot be equivalent. On the other hand, if the set of signature functions for all outputs is comparable, a set of output permutations $\mathcal{P}_o$ can be calculated. Depending on the amount of aliasing, an exact correspondence can be defined. Futhermore, the data can then be used to eliminate unreachable input permutations, by comparing possible correspondences and removing any which are unreachable. Eventually, a smaller set of possible input permutations $\mathcal{P}_i$ remains [8, 50]. Thus, the set of criteria for combinational equivalence can be defined as follows:

1. The number of data inputs and outputs must be equivalent: $|\mathcal{X}| = |I|$, $|\mathcal{Y}| = |O|$.
2. The signature functions of the outputs must be equivalent: $\mathrm{Sig}(\mathcal{Y}) = \mathrm{Sig}(O)$.
    - A set of output permutations $\mathcal{P}_o$ can be found.

- A set of input permutations $\mathcal{P}_i$ can be found.

3. Equivalence is verified through SAT or QBF solving.

In this thesis, the cofactor signature is used to determine I/O correspondence. This signature function strongly demonstrates the effect of each input on the output, through the use of the positive cofactor of each output function. The positive cofactor of a function f_x is defined as the function, with the input x set to "1". For each output, the cardinality of the set of all satisfying input points for the positive cofactor f_{x_i} for each input x_i is calculated, otherwise known as the the satisfy count of each cofactor [8, 50].

Example 4.3.1. Consider one component and two subcircuits, each with two outputs ((f, g) and (x, y) respectively) and four inputs ((i_o, i_1, i_2, i_3) and (a, b, c, d) respectively). The number of inputs and outputs for both subcircuits and the components are identical, thus further testing is possible; they may be equivalent. Given the following function for output $f = ((i_1 \vee i_0) \wedge (i_1 \vee \overline{i_2}) \wedge (i_1 \vee \overline{i_3}) \wedge (\overline{i_0} \vee \overline{i_2} \vee i_3))$, the positive cofactor is $f_{i_o} = (i_1 \wedge (i_1 \vee \overline{i_2}) \wedge (i_1 \vee \overline{i_3}))$. This cofactor can be satisfied for four input points, that is, four sets of input assignments will allow the cofactor to evaluate to "True" (for example, with $(i_1 = 1, i_2 = 1, i_3 = 1)$). This gives the output f a cofactor signature $S(f(i_o)) = 4$, as four possible input assignments exist.

Consider the following signatures for the component

$$S(f(i_0, i_1, i_2, i_3)) = (4, 1, 5, 4), \quad S(g(i_0, i_1, i_2, i_3)) = (3, 2, 1, 3)$$

and for subcircuit A:

$$S(x(a, b, c, d)) = (2, 3, 1, 1), \quad S(y(a, b, c, d)) = (4, 4, 1, 5)$$

The sorted set of signatures for the component is $((1, 4, 4, 5), (1, 2, 3, 3))$ and for subcircuit A is $((1, 4, 4, 5), (1, 1, 2, 3))$. As the set of signatures is not equivalent, subcircuit A is disqualifies from further equivalence testing.

Consider subcircuit B, with the following signatures:

$$S(x(a, b, c, d)) = (2, 1, 3, 3), \quad S(y(a, b, c, d)) = (1, 5, 4, 4)$$

The set of signatures of Subcircuit B is comparable to the component. Inspection of the signatures shows that $f \equiv y, g \equiv x$, giving a exact output permutation. Now, the signature data can be used to eliminate unreachable input permutations. For four inputs, there exist 24 possible input permutations. Using output f and y, eight permutations can be eliminated. For example, it can be seen that $i_o \not\equiv a$. Using output g and x eliminates further

permutations. However, as both outputs contain identical signatures for two inputs i_0, i_3, c, d, the identification of an exact input permutation is impossible. The set of possible permutations is $(f \equiv y, g \equiv x, i_1 \equiv a, i_2 \equiv b \ \{i_0 \equiv c, i_3 \equiv d \ || \ i_o \equiv d, i_3 \equiv c\})$, reducing the search space from 48 to 2 possible permutations.

When additional control wires are defined for the subcircuit, the number of data inputs and outputs must remain identical to those of the component. Furthermore, the signatures for the outputs of the subcircuit must be calculated for all possible permutations of control wires. For a subcircuit representing part of an ALU, with two control wires controlling different functions, four sets of signatures are required. This can be reduced by eliminating impossible control wire assignments. As only one function can run at a time, no signature set is required for the assignment controlWire1 = "1" and controlWire2 = "1" or when both control wires are "0" ; only two signature functions are valid.

Although a vast number of viable signature functions exist, the cofactor signature is a particularly fast and efficient signature function [8, 50]. While a more complex and slower signature function may be optimised to reduce aliasing, and thus can find an exact I/O match more easily, the reduction in search space by using the cofactor signature is sufficient that an exact match is not required. Mohnke and Malik [50] propose the use of an additional break-up signature to reduce aliasing, but again, this was not required for the thesis. The additional computation is more complex than working with a reduced set of possible I/O permutations. While for the worst case scenario, where aliasing occurs for every output and input, the number of possible I/O permutations remains unchanged at $i! \times o!$, the best case promises an exact match. With the circuits used in this thesis, the number of possible permutations was generally within an order of magnitude of $i \times o$.

This method proposes a speedy approach to I/O correspondence, applicable to cases where brute force matching becomes impossible. Once a set of possible output and respective input correspondences is calculated for a subcircuit and library component, the final step is to verify functional equivalence between the two. Several methods for CEC are outlined below.

4.3.3 Satisfiability solving

Satisfiability (SAT) Solving, used in the field of hardware synthesis and verification, is used in this thesis to verify whether a subcircuit is functionally equivalent to a given library component [73]. Using SAT Solvers allows for Combinational Equivalence Checking (CEC) between circuits of the same functionality, but different implementation. As detailed in chapter 3, there exist a multitude of implementations and optimisations for SAT Solving, here, only the basic theory of SAT will be discussed.

Essentially, a SAT Solver determines whether a boolean formula is satisfiable, that is, whether there exists a variable assignment for which the boolean formula becomes true [73]. Thus, the problem of CEC between two circuits is defined as whether there exists an assignment of input variables for which the circuits are not equivalent. If this

problem is not satisfiable, the two circuits are functionally identical. The functionality of the two circuits is expressed as a single boolean formula, which evaluates to '1' only if both circuits are not equivalent. This can be achieved through the use of a miter circuit, which combines the inputs and outputs of two separate circuits [23]. Given subcircuit $c \in \mathcal{C}$ with functionality $\mathcal{Z} = (z_1(\mathcal{X}), z_2(\mathcal{X}), \ldots, z_{|\mathcal{Y}|}(\mathcal{X}))$ and abstract component $\alpha \in \mathcal{L}$ with functionality $\mathcal{S} = (s_1(I), s_2(I), \ldots, s_{|O|}(I))$, the miter circuit can be described as:

$$z_1(\mathcal{X}) \oplus s_1(I)) \vee (z_2(\mathcal{X}) \oplus s_2(I)) \vee \cdots \vee (z_{|\mathcal{Y}|}(\mathcal{X}) \oplus s_{|O|}(I) \tag{2}$$

The miter circuit is constructed as illustrated in Figure 4.8.

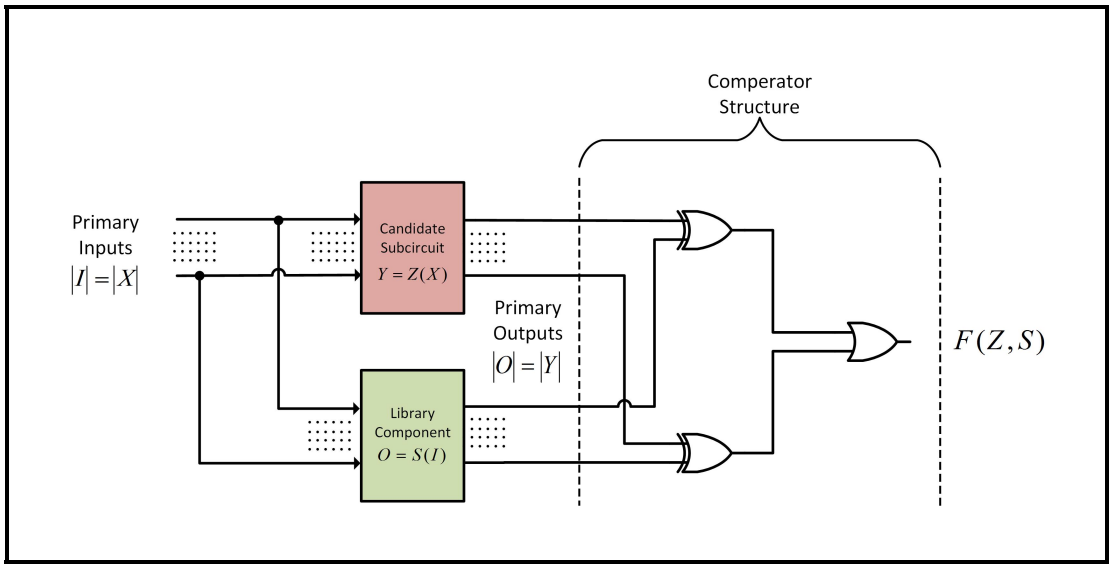


Figure 4.8: SAT Solving

First, the PIs of both circuits are connected, using the correspondence previously calculated, giving a single set of inputs to the boolean formula [23]. Similarity, the POs of both circuit are connected according to the I/O permutation, but using a single XOR gate for each set of outputs. The XOR gate has the unique functionality that it's output is high only if the two inputs to the gate are not identical. As both circuits receive the same input information, this means that a XOR gate evaluates to high only if the functionality of both outputs is inequivalent. It only requires two connected outputs to be non-equivalent to preclude circuit equivalence, thus circuit equivalence can be given only if all XOR gates evaluate to '0'. Hence, all XOR gates are connected to a single OR gate, which evaluates to '1' only if one or more of the outputs of the subcircuit and component are not equivalent. A boolean function can be extracted from this circuit with ease, in particular if the output functions of both circuits are pre-calculated. This boolean function can be formulated as a SAT Problem, where, if a satisfiable input assignment exists for which the output of the miter circuit is '1', the circuits are inequivalent.

Although the SAT Problem is NP-complete [73], with exponential worst case computational complexity $\mathcal{O}(2^n)$, with the use of efficient and optimised solvers, most

problems can be solved in near linear time [32]. Of the three methods outlined in this thesis, SAT solving is the most speedy method for the identification of subcircuit functionality. In general, the number of I/O permutations control the runtime, rather than the number of variables in the boolean formula. Moreover, once an equivalence is found between a subcircuit and a component for a given I/O permutation, no further permutations need to be tested, and it is certain that this component completely represents the functionality of the subcircuit.

A disadvantage of this method is that the entire candidate subcircuit must contain the exact functionality of the component to which it is compared. This punishes errors during candidate subcircuit enumeration, in particular if subcircuits cannot be carved out completely. Furthermore, possible control wires are treated as data inputs. This means either, that the component library must also contain components with control wires, or that candidate subcircuits must not contain control wires. When using control wires during subcircuit enumeration, incomplete subcircuits are extended to include a path to control wires. This means that, for equivalence to be possible, the component library must contain a component with the functionality of the subcircuit and with an additional path to a unknown control wire. Unless a component has been created specifically for this synthesised netlist, it is highly unlikely that exactly this subcircuit is contained within the component library. If instead, candidate subcircuits are created without the use of control wires, less candidate subcircuits are detected, but, generally, more of these can be identified. Depending on the state of the component library, either option can be chosen, that is, creating subcircuits without control wires, or assuming that functional module with control wires is included in the component library .

Thus, SAT solving is a speedy method for establishing equivalence between a candidate subcircuit and a component, but is less than ideal for netlists containing control wires or functionality which is not included in the component library.

4.3.4 Quantified Boolean Formula Satisfiability Solving

From the area of partial design verification comes the idea of using Quantified Boolean Formula (QBF) solving for CEC between two circuits, when the functionality of one circuit is partially unknown. For the equivalence between a candidate subcircuit and a library component, this allows for some some flexibility in the carved out candidate subcircuit; the functionality of the candidate subcircuit does not need to match that of the component completely, except under a specific assignment of control wires [43].

QBF solving is a generalisation of traditional SAT solving [59]. Similarly to SAT, QBF seeks to find variable assignments for which a Boolean Formula is satisfied, but variables are universally or existentially quantified. For a universally quantified variable, rather than testing whether a single variable assignment exist which satisfies the Boolean Formula, all variable assignments must satisfy the Boolean Formula. Existentially quantified variables must provide one or more variable assignments which satisfy the Boolean Formula, as with SAT solving. Li et al. [43] proposes to universally quantify data inputs in CEC, while control wires, for which only a specific variable assignment is required, are existentially quantified. This only requires two levels of quantification, commonly described as the 2QBF Problem. Thus, the CEC Problem for 2QBF can be defined as: exists an assignment to all existentially quantified control wires which

satisfy the Boolean Formula and, for this assignment, do all assignments to universally quantified data inputs satisfy the Boolean Formula.

The functionality of the candidate subcircuit and library component can be expressed as a single boolean formula, which evaluates to "1" only if both circuits are equivalent. Given subcircuit $c \in \mathcal{C}$ with data inputs $\mathcal{X}$, side Inputs $\mathcal{X}_s$, and functionality $\mathcal{Z} = (z_1(\mathcal{X} \cup \mathcal{X}_s), z_2(\mathcal{X} \cup \mathcal{X}_s), \ldots, z_{|\mathcal{Y}|}(\mathcal{X} \cup \mathcal{X}_s))$, and library component $\alpha \in \mathcal{L}$ with functionality $\mathcal{S} = (s_1(I), s_2(I), \ldots, s_{|O|}(I))$, the boolean function can be defined as follows:

$$\exists \mathcal{X}_s,\ \forall \mathcal{X}\ \overline{(z_1(\mathcal{X} \cup \mathcal{X}_s) \oplus s_1(I))} \wedge \overline{(z_2(\mathcal{X} \cup \mathcal{X}_s) \oplus s_2(I))} \wedge \cdots \wedge \overline{(z_{|\mathcal{Y}|}(\mathcal{X} \cup \mathcal{X}_s) \oplus s_{|O|}(I))}$$

If this QBF can be satisfied when the output of the miter circuit is '1', the circuits are equivalent for a specific assignment of control wires.

Again, a miter circuit is formed by combining the inputs and outputs of both the subcircuit and library component, using the precomputed I/O permutations, as illustrated in Figure 4.9.

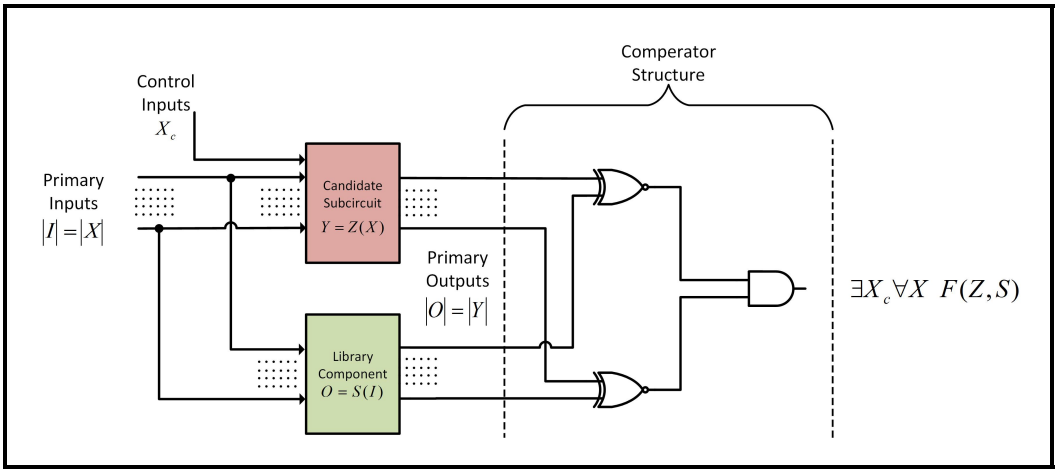


Figure 4.9: QBF Solving

Instead of finding unsatisfiability for circuit inequivalence, QBF solving seeks satisfiability for circuit equivalence. Consequently, the outputs of both circuits are combined using XNOR cells, which evaluate to high only if both inputs are identical, rather than XOR cells. Hence, for two equivalent circuits, every output must be identical in function, and thus the output of all XNOR cells must evaluate to '1', which can be verified with a single AND cell connecting all XNOR outputs.

QBF holds a distinct advantage over SAT Solving, in that it allows for a margin of uncertainty in the functionality of the candidate subcircuit. Especially when acknowledging the difficulty of generating perfect candidate subcircuits during subcircuit enumeration, this is a significant advantage. Furthermore, a match found between a subcircuit and component for a given I/O permutation does not disqualify all other I/O permutations, as a new permutation may lead to a new match. There is a good possibility that a subcircuit may match more than one library components. This increases runtime, but also the information gained regarding each candidate subcircuit.

By definition, solving a QBF instance is computationally more complex than solving a SAT instance, being PSPACE complete rather than NP-complete [59]. However, limiting the problem to two levels of quantification, and the use of efficient solvers generally avoids this worst case complexity. Once again, the runtime is influenced more by the possible number of I/O permutations, and number of control wires, than the number of inputs to both circuits. This further highlights the importance of an efficient approach to I/O correspondence, as all I/O permutations are tested. It must be noted, that if no control wires are defined, QBF solving encounters much the same difficulties as SAT solving, with a worse runtime.

4.3.5 Satisfiability modulo Theories solving

The success of both SAT and QBF Solving is in part based on the completeness of the component library. In particular, non-standard functions may not be contained within the component library, and thus, any candidate subcircuits implementing this functionality cannot be identified. Moreover, all combinations of sizes for input and output bit vectors must be contained in the library. To reduce the effort of creating a complete component library, Gascon et al. [28] proposes the use of Satisfiability module Theories (SMT) based templates instead of concrete implementations to solve the CEC Problem.

Templating describes the idea to use predefined syntactic templates to generate a specific instance of an implementation during runtime [1]. For example, for CEC, consider a candidate subcircuit with 16 inputs and 8 outputs, implementing an Adder. Instead of identifying this circuit by comparing it to a specific library component which implements an Adder, a template can be filled with the specification of an Adder during runtime, including the required number of inputs and outputs. In order to create a solvable instance, information regarding the candidate subcircuit is extracted, including a SMT compatible representation of the circuit and a statement describing the quantification of data and control inputs. Moreover, further information regarding the component is generated based on the characteristics of the subcircuit, such as the size of the component inputs and outputs. A template based on SMT lends itself for this particularly well, as it significantly reduces the effort to describe a component.

SMT extends traditional SAT solving to other domain theories outside of the boolean domain [74]. Where SAT Solving is based purely on the boolean functionality of a component, SMT allows for the specification of functionality with bit vectors, arrays or arithmetic [74]. Paired with the idea of using a template rather than an instance, a template library spanning a wide range of functions can be created. In this thesis, the theory of bit vectors is used to create instances, limiting the functionality of the library to combinational logic.

Given an abstract template $\mathcal{T}$ with SMT functionality $\mathcal{S}$, a candidate subcircuit c_k, with functionality $\mathcal{Z}$, can be mapped to the template, such that a SMT instance $\mathcal{T}_{c_k}$ is created. If this instance can be solved for all data inputs, and for a given assignemnt of control inputs, the functionality of the candidate subcircuit if equivilant to the template functionality:

$$\mathcal{Z} \equiv \mathcal{S} \iff \exists \mathcal{I}_c \; \forall \mathcal{I} \; \mathcal{T}_{c_k} : c_k \mapsto \mathcal{T} \text{ is satisfiable} \tag{5}$$

A SMT template may contain keywords for the circuit representation, circuit inputs and outputs, and a component representation. During runtime, these keywords are replaced with the corresponding data to create a SMT instance. A more complex template representation includes the information required to generate all permutations of inputs and outputs. Gascon et al. [28] proposes a signature based method to reduce the number of I/O permutations required. In this thesis, this option was not further pursued, as the current set-up did not lead to a significantly longer runtime than using QBF Solving with signatures. In the worst case, SMT solving is again PSPACE Complete, but due to the optimisations of the SMT Solver, the worst-case is not usually encountered.

As mentioned above, a single SMT template can represent a significant number of implementations and bit (sizes) for a given functionality. For example, if a 17 bit Adder is found as a candidate subcircuit, this specific component will also need to be contained within the component library in order to identify the subcircuit, while a single template for Adders is sufficient when using SMT based templates. Furthermore, using SMT solving once again allows for the existential quantification of control wires, which provides some error tolerance to the candidate subcircuit enumeration procedure.

One of the main weaknesses of this method is the difficulty of creating the template library, in particular with respect to the representation of more complex structures. Including a module in a netlist based component library is as simple as synthesising the netlist from a Hardware Description Language (HDL) description. Often, even complex modules already exist as 3PIP with a HDL description. This is not the case for SMT based templates, and while the number of required templates in a template library is significantly reducing by expanding the functional space which each template represents, each template nevertheless needs to be created by the user. In particular, creating more complex modules which cannot easily be represented with bit vectors, not only requires a detailed understanding of the SMT language, but become of similar complexity as a full HDL description, negating the advantages of using a SMT representation.

4.4 Post Processing

The above methods provide a sound solution to the subcircuit identification problem. However, while a set of candidate words may now be identified, post processing is required to provide a final output. As each node of the netlist may be contained in more than one candidate subcircuit, after functional identification, further steps must be taken to eliminate overlapping. Furthermore, uncovered sections of the circuit can be further analysed.

The coverage of a netlist can be calculated by identifying all nodes not included in any identified candidate subcircuit. If uncovered nodes remain, it is possible that these may form further feasible candidate subcircuits. Consequently, all connected uncovered areas are examined, and if they form feasible subcircuits, identification should be attempted using CEC.

More interesting is the problem of extracting the best resolution of subcircuits for the entire netlist. When no further uncovered nodes can be processed, all matches must be analysed with regards to overlap. Subramanyan et al. [65] propose the use of a resolution function which maximises the coverage and minimises the number of subcircuits used to achieve this, two solutions using a binary integer linear program are presented. Here a more simple approach is used.

First, smaller circuits which are completely contained within larger subcircuits are analysed. If they are identified for same set of control wire assignments, the smaller circuits is discarded. For example, a set of nodes may be identified as an Adder / Subtractor (addsub) circuit, with a subset of the nodes identified as an adder. If the circuits were identified for the same control wire permutation, the smaller circuit is no longer required, as it provides less information. On the other hand, if two candidate subcircuits are identified for separate control wire assignments, the subcircuits may be grouped into a larger module. For example, two separately identified subcircuits representing an adder and a subtractor with different control wire assignments, may be grouped to an addsub circuit. This reduces the total number of submodules.

Now, a resolution function can identify all sets of disjoint circuits, and depending on the criteria, one or more of these sets can be picked to represent a high level description of the netlist. In this thesis, more emphasis is put on the largest coverage. Although a smaller set of nodes, and thus a smaller coverage may provide a more high-level description, an identified cell provides more information, even if the containing subcircuit is less high-level. For example, when three subcircuits are identified, and two of these cover a larger subset of nodes, yet provide less high-level information, these are preferred to a more complex subcircuit, covering a smaller set of nodes. In this thesis, a case like this will generally result in two possible solutions, in line with the idea that information should not be disregarded without good cause.

Chapter 5: Results

This project is implemented using exclusivity open-source software. It is written in python 3.4, extended with libraries for graph manipulation [25], boolean algebra [25] and cluster detection [3, 38]. Circuits are synthesises using the open source framework Qflow 1.1 [26], with yosys for front end synthesis [77]. Both the "NanGate 45nm Open Cell Library" (nangate) [52], and the "Oklahoma State University Cell Library 035" (osu035) [69] cell libraries are used for synthesis. For SAT Solving, the minisat solver [27] is used, QBFs are solved with DepQBF [44], and Microsofts z3 [20] is used to solve SMT instances. To better understand the methods and ideas presented in this thesis, a small set of results is analysed here.

5.1 Candidate Subcircuit Enumeration

The project was tested for the following designs, as detailed in Table 5.1. The aes is a 32-bit aes encryption core, and the S-Box is a functional submodule of the aes. The alu is a 8-bit parallel ALU, implementing an addition, addition with carry, subtraction, subtraction with carry, and Boolean AND, XOR and OR. The design mul is a 4 bit multiplier, repeated eight times, to give additional depth. Both design c71481 and design c6288 are included in the original ISCAS-85 combinational Benchmarks, implementing a 4-bit function generator, and a 16-bit multiplier respectively. The IDEA round is an 4-bit round module of the IDEA encryption algorithm, containing most notably four multipliers. Design ecc contains a small masked error correction circuit, with a single masking bit. Furthermore, to allow for comparison of the effect of input bit size on runtimes and data found, a set of differently sized Adder / Subtractor (addsub) was included.

Table 5.2 contains the number of words found per design, both structural and functional, including the number of words found before and after a feasibility check. It also illustrates the number of source words which remained after the functional and structural words are merged and refined with regards to overlap and subsets. It can be seen that both structural and functional heuristics are adequate in identifying possible words, but that, depending on the design, many of these were not within the required criteria, and thus removed. Depending on the structure of the design, more words were identified using functional characteristics, but of these a larger amount was discarded. Commonly, many of the words found using structural heuristics were also included in the functional identification. In particular for smaller designs, the number of words which are used for the generation of the data path is very limited, as many of the identified words overlap.

Design Name	Nr. of Gates	Nr. Nets	Description	Cell Library
ecc	92	142	masked 7-bit ecc	osu035
c71481	122	191	ISCAS-85 4-bit Function Generator	osu035
alu	163	316	8-bit alu with 7 functions	nangate
mul4	688	1352	4-bit multiplier, repeated x8	osu035
IDEA round	896	1608	4-bit IDEA round module	osu035
S-Box	1006	1896	8-bit masked S-Box	osu035
c6288	2437	4507	ISCAS-85 16-bit Multiplier	nangate
aes	5265	10008	32-bit aes encryption core	nangate
addsub4	62	97	4-bit addsub	osu035
addsub6	107	171	6-bit addsub	osu035
addsub8	155	251	8-bit addsub	osu035
addsub16	357	589	16-bit addsub	osu035

Table 5.1: Design Information

Design Name	Structural	t (sec)	Feasible	t (sec)	Functional	t (sec)	Feasible	t (sec)	Source	t (sec)
ecc	2	0	2	0	33	2	26	4	4	0
c74181	5	0	5	0	63	3	50	7	12	0
alu	4	0	4	0	28	3	20	1	14	1
mul4	63	0	0	0	27	14	0	0	0	0
IDEA round	59	0	4	3	314	36	22	8	12	4
S-Box	56	0	5	3	133	48	20	14	10	0
C6288	120	0	0	3	210	75	10	124	1	0
aes	482	1	78	361	115	384	69	642	76	0
addsub4	0	0	0	0	16	1	7	1	1	0
addsub6	0	0	0	0	36	2	3	1	2	0
addsub8	0	0	0	0	68	2	4	1	1	0
addsub16	0	0	0	0	137	5	16	11	1	0

Table 5.2: Word Identification

Considering the two multipliers is of particular interest. Although words are identified using both structural and functional heuristics, almost none of these can be declared as source words. This is caused by the exceedingly regular structure of multipliers, which are commonly implemented using a lattice of half and full adders. These all present in extremely similar cuts and shapes, and while many candidate words are found, the algorithm cannot detect which of these form actual words based on distance or hierarchical checks. For the addsub designs, no words were identified using structural characteristics, while the number of candidate functional words increases with the input size, and thus number of nodes in the graph. Yet after refinement, only few of these words remain. Both of these outcomes are again caused by the regular structure of the design, which means that many overlapping words are identified.

The runtime for structural word identification is insignificant, even for larger designs, which is in line with the computational complexity of $\mathcal{O}(n)$. Similarly, the runtime for cut calculation is exponential ($\mathcal{O}(n^{4k}k)$). However, in larger designs, the most significant impact on the runtime of word identification is caused by the feasibility checks on each word, not only as certain criteria must apply for every word (as detailed in subsubsection 4.1.1.4), but also because the set of words must be reduced, so that complete subsets and overlapping words are removed.

Once a set of source words has been declared, the word propagation algorithm identifies further words and creates a hierarchy of words. In general, word propagation is only possible with the existence of control wires, either by design, such as control inputs, or through optimisation, as suggested in [43]. Here, both methods are used; with differing results, as illustrated in Table 5.3. For example, the alu has a high number of control wires, but due to the structure and limited depth, very few candidate words were found, and of these, only some could be propagated. Contrarily, the IDEA round contains no control inputs, but for each candidate word a high number of possible control wires introduced by optimisation was found, and thus several words could be propagated. This means that the number of control wire will not always impact the number of propagated words, rather this is dependent on the structure of the design. However, it is true that a lack of control inputs and control wires will always mean that the a word cannot be propagated, as is the case for the small mul4 design.

Design Name	Input	Output	Control	Candidate	Propagated	t (sec)	Nodes D	Edges D
ecc	2	1	1	170	5	3	22	13
C74181	3	1	2	235	4	3	22	56
alu	2	1	11	49	4	3	26	22
mul4	2	1	0	512	0	52	3	2
IDEA round	6	1	0	140	11	5	37	169
aes	6	1	1	137	23	29674	137	26
addsub4	2	1	1	5	4	0	9	11
addsub6	2	1	1	6	4	2	10	14
addsub8	2	1	1	5	2	24	8	8

Table 5.3: Word Propagation

Table 5.3 summarises the results for the word propagation algorithm, for the sake of completeness, the number of input and output words, as well as control wire bits are given. On the right, the number of nodes and number of edges of the data path are recorded. For both multipliers, during word propagation, the extremely regular structure of the designs results in many forward words, which presents in extreme runtime, and a multitude of candidate words. For example, a input word to a 8-bit multiplier can have more than 50,000 possible forward word and with increasing bit size, this number has super multiplicative growth. Even for the small mul4 design, more than 500 candidate words are found, for the c6288, the generation of forward words requires extreme runtimes and memory. Furthermore, due to the fact that no control wires are included in a regular multiplier, none of these words can be propagated, and thus the data path is extremely small, resulting in the worst-case for word propagation.

The aes is also of interest; the runtime of the word propagation algorithm is extremely high at $\approx$ 8 hours. This is caused by the high interconnectivity of the contained S-Boxes; during the forward and backward propagation steps, a large number of words are found, which are then disregarded as they commonly overlap. For example, for a single 8-bit word within the aes, more than 20,000 forward words can be found, of these, commonly less than 10 are disjoint. This reiterates the idea that the runtime for the word propagation can be difficult to estimate, and is dependent on the interconnectivity of the design.

Finally, the family of addsubs showcases a nearly ideal case of word propagation; a

similar number of words are found and propagated for each input size, and a similar sized and structured data path is created. As the structure of an addsub circuit should remain identical, independent of input size, this is the ideal outcome.

In order to identify feasible candidate subcircuit, a set of boundary words is generated from the data path. If only a small number of edges, compared to the number of words, exist within the data path, the set of possible boundary words is very small. For a larger number of edges, the possibilities of input word combinations become very large. As already discussed, when a single word, with a ancestors in the data path, acts as the output to a functional module which has a maximum of c input words, $\sum_{i=1}^{c} \frac{a!}{i!(a-i)!}$ combinations of input words are possible. Thus, more edges in the data path, and thus more ancestors, can lead to incredibly high numbers of boundary word sets. On the other hand, when the word propagation algorithm cannot propagate data from the PIs to the POs of the design, and structural analysis cannot find further connections between words, few edges in the data path will exist, resulting in few boundary word sets. This is the case in both the mul4 and the aes designs; although words were propagated, many words within the data path remain unconnected. Subsequently, Table 5.4 only includes designs with a high set of edges in the data path. Especially as only few of the boundary words lead to feasible subcircuits, the quality of the data path is significant.

Design Name	Boundary Word Sets	t (sec)	Feasible Subcircuits	t (sec)
ecc	173	1	11	4
C74181	1347	4	10	4
alu	501	2	8	6
IDEA round	24150	297	0	461
addsub4	67	0	4	0
addsub6	85	0	7	0
addsub8	30	0	4	0

Table 5.4: Subcircuit Generation

Once again the number of control wires directly affects the outcome. As previously discussed, a feasible candidate subcircuit must fulfil a set of criteria, one of these being that every node must have the required amount of inputs. Control inputs allow the subcircuit enumeration algorithm to extend possible subcircuits until they become feasible. For the IDEA round design, the high number of edges in the data path of the IDEA round results in a large number of possible boundary words. Both the creation and the verification of these is time intensive. However, as none of these are feasible and, due to the the lack of control wires, none can be extended, no feasible subcircuits can be identified. The addsub designs again illustrate the ideal case: the clear data path results in a small number of possible boundary words, of which many can either be extended, or are already feasible.

From these results, it can be seen that using a data path structure to identify candidate subcircuits is not always ideal, in particular for larger designs, and those with no inherent control inputs or wires. Often, a low quality data path will result in a large number of boundary words, and the resulting subcircuits require large runtimes for verification. Furthermore, for highly interconnected designs, or those with a extremely

regular structure, identifying words and the relation between these can be difficult and time intensive.

5.2 Circuit Clustering

Circuit clustering addresses the problems which arise when too few edges exist in the data path, too many boundary words are identified, or too few candidate subcircuits are identified. As mentioned, circuit clustering cannot by itself be used to identify functional modules, as the boundaries are too vague. However it can be used to support the identification of feasible words, feasible boundary word sets, and feasible candidate subcircuits.

In this project, four separate clustering algorithms, based on three different methodologies are tested: the Louvain method, the MCL algorithm, a optimised MCL algorithm, and the Girvan-Newman algorithm. As expected, each of the clustering algorithms provides a different number of clusters for a design, as each is based on a different clustering heuristic. Results for a set of circuits are shown in Table 5.5. The Girvan-Newman algorithm (GN) supplies a set of clusters, the number of cluster sets is shown under "GN Number", while the number of clusters in the cluster set with the highest modularity is recorded under "GN best".

Design Name	Louvain	t (sec)	MCL (opt.)	t (sec)	MCL	t (sec)	GN best	GN Number	t (sec)
alu	9	0	4	0	11	0	9	7	7
IDEA round	11	1	4	1	9	54	15	13	692
S-box	10	0	5	1	5	5	13	11	1897
aes	10	2	16	4	26	731	n.a.	n.a.	n.a.

Table 5.5: Clustering Data

The runtime assessment generally holds true, the Louvain method commonly requires < 1 sec, as expected with a computational complexity of $\mathcal{O}(n \log n)$. When using MCL, the optimised version $\mathcal{O}(nk^2)$ runs significantly faster than the manual implementation $\mathcal{O}(n^3)$. Finally, as expected, the Girvan-Newman algorithm is unsuitable for larger designs, as for each set of clusters, a computational complexity of $\mathcal{O}(n^3)$ is required, so that, for the aes design, the algorithm does not complete.

For designs with a clear structure, it was found that large clusters generally overlap functional modules, but by no means provide a clear enough boundary for functional identification. An example is illustrated in Figure 5.1, which shows the clusters found when running MCL on a single round of the IDEA round; the four main functional modules can be clearly identified. In comparison a MCL of an S-Box (Figure 5.2), shows no functional modules. In fact, most cells were assigned to a single cluster, only a few outliers were considered separate enough to warrant a different cluster. This is in line with the structure of the S-Box, which does not contain easily identifiable functional submodules.

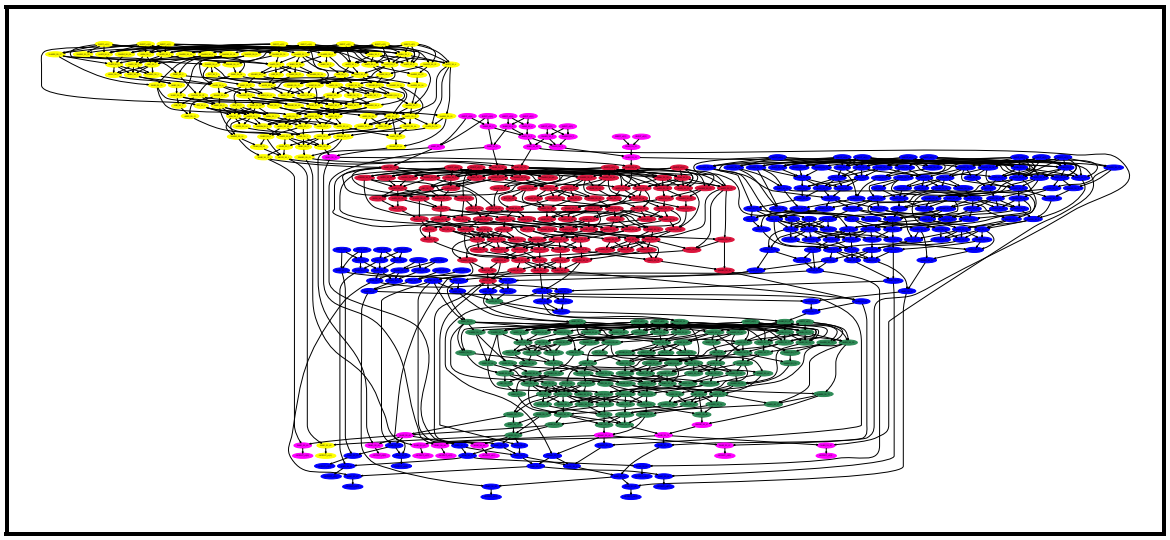


Figure 5.1: Cluster Example of IDEA round module using MCL Algorithm

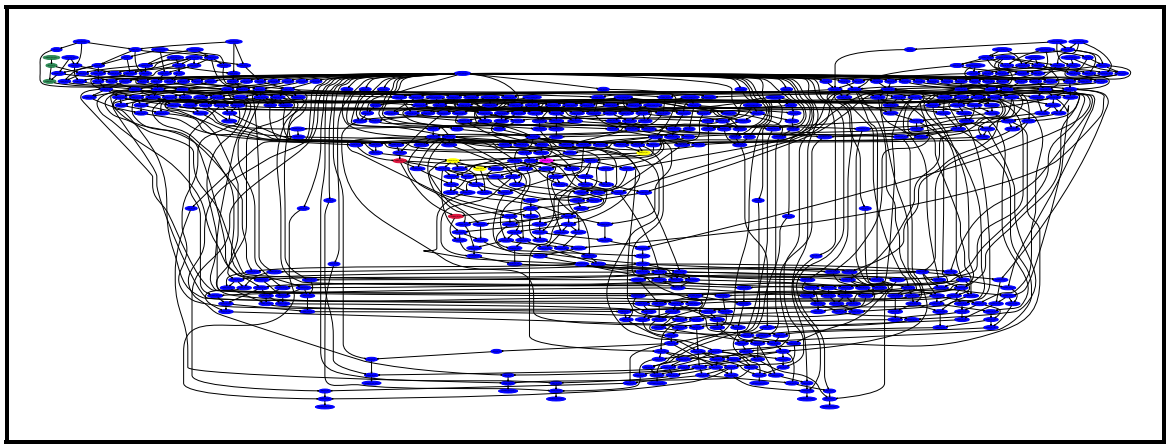


Figure 5.2: Cluster Example of S-Box using MCL Algorithm

When comparing the same designs with the Louvain method, a better output is created for the S-Box (Figure 5.4), while the round module shows that three of the four main functional modules were split (Figure 5.3). Consequently, the Louvain method is preferred for finding differences for more closely interconnected designs, but does this at the cost of dividing larger clusters.

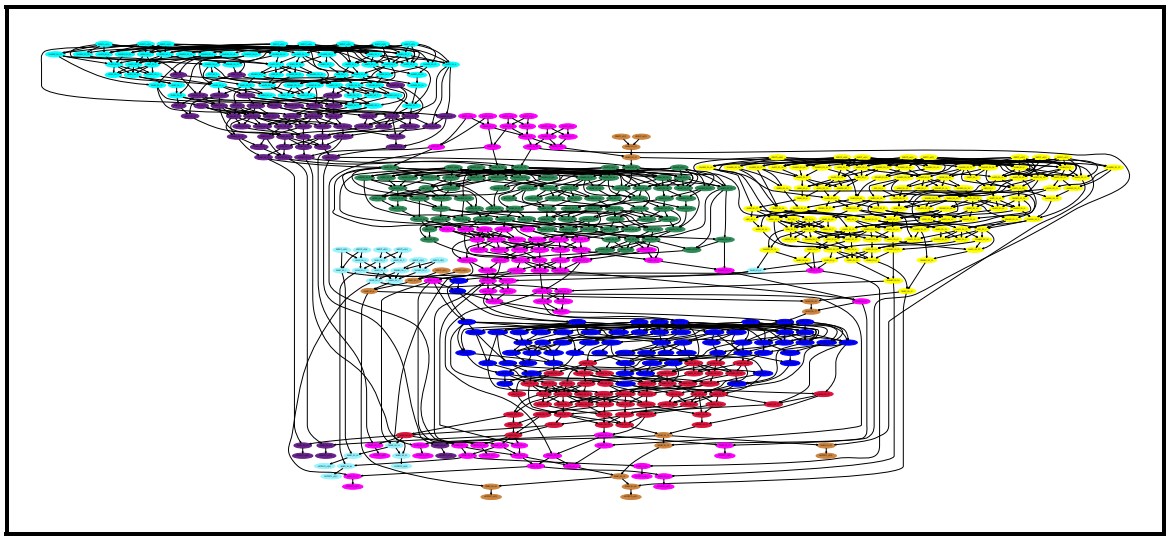


Figure 5.3: Cluster Example of IDEA round module using Louvain Method

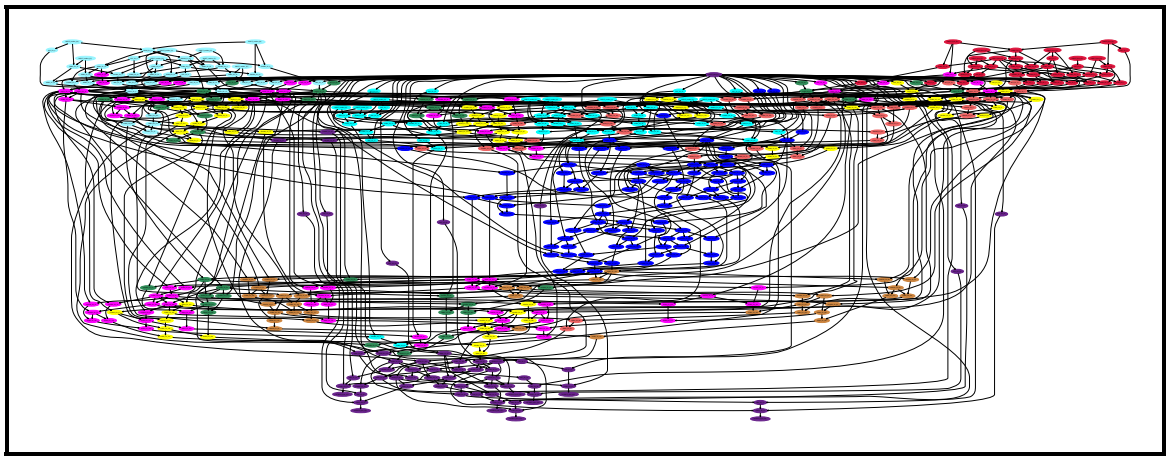


Figure 5.4: Cluster Example of S-Box using Louvain Method

All clustering methods were used in the thesis, with smaller clusters generally removed, depending on the use case. For example, to support the finding of feasible words during word identification, clusters of smaller size than the word itself are ineffectual, only words which are completely included in a cluster significantly larger than the word itself should be given preference. The same holds true when searching for boundary word sets partly or completely contained within a cluster, for subcircuit enumeration.

5.3 Subcircuit Identification

To better study and understand the differences between SAT, QBF and SMT solving, a set of circuits were matched to a component and template library, including two

possible subcircuits of the ecc design, a set of addsub designs, and an alu. The results are summarised in Table 5.6,

Circuit	Nr. SCs	Coverage	Functionality	Id. by SAT	t (I/O)	t total (sec)	Id. by QBF	t (I/O)	t total (sec)	Id. by SMT	t (sec)
ecc	1	64%	xnor/xor	n.a.	n.a.	n.a.	all	3	4	all	2
ecc	1	31%	xnor/xor	n.a.	n.a.	n.a.	all	1	1	all	1
addsub4	3	100%	add, sub, nand	nand	1	1	all	4	6	all	2
addsub6	3	100%	add, sub, nand	nand	2	2	all	13	16	all	12
addsub8	3	100%	add, sub, nand	nand	253	296	all	581	639	all	316
alu	7	100%	add, adc, sub, sdc, and, xor, or	n.a.	n.a.	n.a.	all	8543	9361	not sbc, adc	505

Table 5.6: Subcircuit Identification

First and foremost, the SAT Solver is unable to solve all subcircuits, those with control wire inputs cannot be solved. For subcircuits without control wires, SAT was consistently more efficient than QBF, even considering the extra subcircuits solved by QBF. When compared to SMT, both methods suffer from the need to calculate an I/O correspondence, this commonly requires more than 90% of the runtime. Even though a signature based approach is already significantly more efficient than using a brute force approach, the overhead still has extreme impact on the runtime. With increasing I/O bit size, the runtime increases exponentially. The SMT solver is generally less influenced by I/O bit size, as it only requires a linear number of further input variables for each additional bit.

Although based on the runtime evaluation, SMT solving seems like an obvious choice for CEC, the considerable effort required of the user to create the template library must be considered. Although this is a one time effort, adding new circuits to a template library is far more complex than including a new design within a component library. Not only is irregular structure often complex to template, but describing it with specific theories such as bit vectors can be difficult. For example, no add with carry (adc) or subtract with carry (sbc) templates were created, and thus these functions cannot be identified in the alu. For the ecc and addsub circuits, both the QBF and SMT Solver were able to identify all functionality, as it is based only on bit vector logic.

While a functional identification of candidate subcircuits using CEC can require a considerable runtime, as long as the appropriate template is contained within the library, the instance will be solved. Whether SMT or QBF Solving is preferred, depends entirely on the use case, including the input and output bit sizes, and the state of the component library. In general, QBF solving will give more flexibility and a broader scope of functionality, while SMT solving will provide a quicker solution, but may not be able to identify more complex structures.

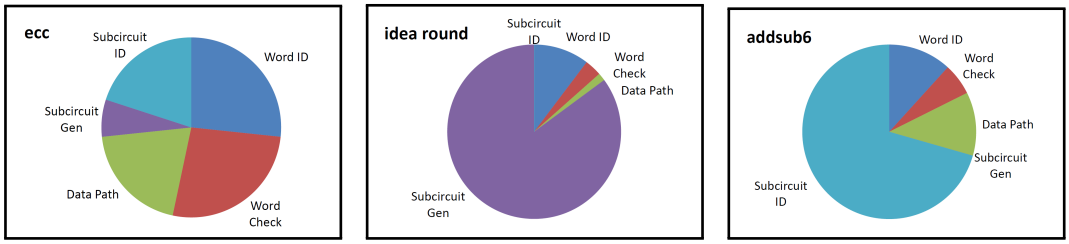


Figure 5.5: Runtime comparison

Finally, an overview of the runtimes for three simple designs, the ecc, a 6-bit addsub and the IDEA round is given in Figure 5.5. It can be seen that no absolute statement can be made regarding which part of this project impacts the runtimes most, as all runtimes are heavily dependant on the individual structure of the design, as well as the input bit size. For larger designs, the subcircuit generation is commonly the lesser part, with the signature calculation and I/O matching contributing most significantly to the runtime. However, as with the aes circuit, word propagation can cause unforeseen runtime complexity.

Chapter 6: Conclusion

This thesis provides an overview of possible methods and algorithms to completely and automatically reverse engineer the high level structure of an unknown synthesised netlist. First, candidate subcircuits are generated. By identifying words using both structural and functional characteristics, and refining them based on a set of criteria, the netlist can be described on a word level basis. Through the use of word propagation, and structural heuristics to find a hierarchy between the identified words, a data path can be re-established. Candidate subcircuits are found by using words contained within the data path as boundaries. Circuit clustering supports these methods by identifying possible functional module boundaries.

Functional identification of extracted candidate subcircuits can be done using SAT, QBF or SMT based solving. This requires a I/O correspondence; instead of a brute force approach, signatures are used to reduce the set of possible permutations. A comparison of the capabilities of SAT, QBF and SMT solving are given, in particular with regards to the required input, such as control wires and subcircuit definition.

A strong focus is placed on implementation and feasibility. The results show that while the proposed methods are capable of reverse engineering many combinational netlist, in particular the method for extraction of feasible functional modules must be refined in the future. Better integration of clustering information is one possibility, functional identification of circuit clusters based on fuzzy matching methods may provide another. Furthermore, while QBF Solving allows for the identification of candidate submodule, if the corresponding functionality is contained within the component library, runtime is an issue, in particular with regards to I/O matching. New methods for faster signature computation should be explored. SMT provides a faster and less I/O size dependant method, but more complex structures cannot easily be included in the template library. In the future, an effort should be made to extend the template library, and more options for the used theories should be explored. In particular, the theory over arithmetic functions may provide an extension to the search space.

This project does not provide a direct method for Hardware Trojan (HT) detection, but instead allows for better understanding of the netlist in question, supporting HT detection. Furthermore, by providing a method for reverse engineering, in the future it may be possible to find countermeasures for IP theft, for example through obfuscation. These options should be further explored in future research.

Acronyms

3PIP	Third Party Intellectual Property
ALU	arithmetic logic unit
AOI	And-Or-Inverter
BDD	Binary Decision Diagram
BFS	breadth-first search
CEC	Combinational Equivalence Checking
DAG	directed acylic graph
DFS	depth-first search
FSM	Final State Machine
HDL	Hardware Description Language
HT	Hardware Trojan
I/O	Input/Output
IC	Integrated Circuit
IP	Intellectual Property
LSB	least significant bit
LTL	Linear Temporal Logic
MCL	Markov Clustering
OAI	Or-And-Inverter
P-equivalence	permutation equivalence
P-representative	permutation independent representative
PI	Primary Input
PO	Primary Output
QBF	Quantified Boolean Formula

RC Adder	Ripple-Carry Adder
ROBDD	Reduced Ordered Binary Decision Diagram
RTL	Register Transfer Level
SAT	Satisfiability
SMT	Satisfiability module Theories
VLSI	Very-Large-Scale Integration

Bibliography

[1] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit Seshia, Rajdeep Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 1–8. IEEE, October 2013. URL `http://dx.doi.org/10.1109/fmcad.2013.6679385`.

[2] N. Anquetil and T. C. Lethbridge. Comparative study of clustering algorithms and abstract representations for software remodularisation. *IEE Proceedings - Software*, 150(3):185–201, June 2003. ISSN 1462-5970.

[3] Thomas Aynaud. Community detection for networkx's documentation. Python Package. URL `http://perso.crans.org/aynaud/communities/index.html`.

[4] M. Banga and M.S. Hsiao. Trusted rtl: Trojan detection methodology in pre-silicon designs. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 56–59, June 2010.

[5] Chongxi Bao, D. Forte, and A. Srivastava. On application of one-class SVM to reverse engineering-based hardware trojan detection. In *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, pages 47–54, March 2014.

[6] Chongxi Bao, D. Forte, and A. Srivastava. On reverse engineering-based hardware trojan detection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1):49–57, January 2016. ISSN 0278-0070.

[7] R. Bazylevych and R. Burtnyk. Algorithms for software clustering and modularization. In *Scientific and Technical Conference "Computer Sciences and Information Technologies" (CSIT), 2015 Xth International*, pages 30–33, September 2015.

[8] Luca Benini and Giovanni De Micheli. A survey of boolean matching techniques for library binding. *ACM Trans. Des. Autom. Electron. Syst.*, 2(3):193–226, July 1997. ISSN 1084-4309.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10:10008, October 2008.

[10] Vincent Blondel. The louvain method for community detection in large networks, March 2011. URL `https://perso.uclouvain.be/vincent.blondel/research/louvain.html`.

[11] Randal E. Bryant. Symbolic simulation-techniques and applications. In *In DAC*, pages 517–521, 1990.

[12] J.R. Burch and D.E. Long. Efficient boolean function matching. In *Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers., 1992 IEEE/ACM International Conference on*, pages 408–411, November 1992.

[13] B. Cakir and S. Malik. Hardware trojan detection for gate-level ICs using signal correlation based clustering. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pages 471–476, March 2015.

[14] S. Chatterjee, A. Mishchenko, R.K. Brayton, Xinning Wang, and T. Kam. Technology mapping with boolean matching, supergates and choices. Technical report, ERL Technical Report, EECS Dept., UC Berkeley, Mar 2005.

[15] S. Chatterjee, A. Mishchenko, R.K. Brayton, Xinning Wang, and T. Kam. Reducing structural bias in technology mapping. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(12):2894–2903, Dec 2006. ISSN 0278-0070.

[16] A. Chowdhary, S. Kale, P.K. Saripella, N.K. Sehgal, and R.K. Gupta. Extraction of functional regularity in datapath circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(9):1279–1296, September 1999. ISSN 0278-0070.

[17] J. Cong and Sung Kyu Lim. Edge separability-based circuit clustering with application to multilevel circuit partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(3):346–357, March 2004. ISSN 0278-0070.

[18] J. Cong, L. Hagen, and A. Kahng. Random walks for circuit clustering. In *ASIC Conference and Exhibit, 1991. Proceedings., Fourth Annual IEEE International*, pages P14–2/1–4, September 1991.

[19] Jason Cong, Chang Wu, and Yuzheng Ding. Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution. In *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, FPGA '99, pages 29–35, New York, NY, USA, 1999. ACM. ISBN 1-58113-088-0.

[20] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78799-2, 978-3-540-78799-0. URL `http://dl.acm.org/citation.cfm?id=1792734.1792766`.

[21] D. Debnath and T. Sasao. Fast boolean matching under permutation using representative. In *Design Automation Conference, 1999. Proceedings of the ASP-DAC '99. Asia and South Pacific*, pages 359–362vol.1, January 1999.

[22] Debatosh Debnath and Tsutomu Sasao. Fast boolean matching under permutation by efficient computation of canonical form. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 87(12):3134–3140, 2004.

[23] S. Disch and C. Scholl. Combinational equivalence checking using incremental SAT solving, output ordering, and resets. In *Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific*, pages 938–943, January 2007.

[24] T. Doom, J. White, A. Wojcik, and G. Chisholm. Identifying high-level components in combinational circuits. In *VLSI, 1998. Proceedings of the 8th Great Lakes Symposium on*, pages 313–318, February 1998.

[25] Chris Drake. Pyeda: Data structures and algorithms for electronic design automation. In *Proceedings of the 14th Python in Science Conference (SciPy 2015)*, pages 26–31, August 2015.

[26] Tim Edwards. URL `http://opencircuitdesign.com/qflow/`.

[27] N. Een and N. Sörensson. Minisat v1.13 - a SAT solver with conflict-clause minimization, system description for the SAT competition, 2005.

[28] A. Gascon, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanovic, and S. Malik. Template-based circuit understanding. In *Formal Methods in Computer-Aided Design (FMCAD), 2014*, pages 83–90, Oct 2014.

[29] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[30] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker. Equivalence checking of partial designs using dependency quantified boolean formulae. In *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pages 396–403, October 2013.

[31] E.I. Goldberg, M.K. Prasad, and R.K. Brayton. Using sat for combinational equivalence checking. In *Design, Automation and Test in Europe, 2001. Conference and Exhibition 2001. Proceedings*, pages 114–121, 2001.

[32] Carla P. Gomes, Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *FAI*, 2008.

[33] Z. Guo, M. Tehranipoor, D. Forte, and J. Di. Investigation of obfuscation-based anti-reverse engineering for printed circuit boards. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6, June 2015.

[34] Lars Hagen and Andrew B. Kahng. A new approach to effective circuit clustering. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '92, pages 422–427, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. ISBN 0-89791-540-2. URL `http://dl.acm.org/citation.cfm?id=304032.304146`.

[35] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *Design Test of Computers, IEEE*, 16(3):72–80, 1999. ISSN 0740-7475.

[36] Clemens Helfmeier, Dmitry Nedospasov, Christopher Tarnovsky, Jan Starbug Krissler, Christian Boit, and Jean-Pierre Seifert. Breaking and entering through the silicon. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 733–744, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9.

[37] U. Hinsberger and R. Kolla. Boolean matching for large libraries. In *Design Automation Conference, 1998. Proceedings*, pages 206–211, June 1998.

[38] Kazem Jahanbakhsh. Community detection in social networks, October 2010. URL `http://www.kazemjahanbakhsh.com/codes/cmty.html`.

[39] J. H. Jahnke. Reverse engineering software architecture using rough clusters. In *Fuzzy Information, 2004. Processing NAFIPS '04. IEEE Annual Meeting of the*, volume 1, pages 4–9 Vol.1, June 2004.

[40] H. Katebi and I.L. Markov. Large-scale boolean matching. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 771–776, March 2010.

[41] Wenchao Li, A. Forin, and S.A. Seshia. Scalable specification mining for verification and diagnosis. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 755–760, June 2010.

[42] Wenchao Li, Z. Wasson, and S.A. Seshia. Reverse engineering circuits using behavioral pattern mining. In *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*, pages 83–88, June 2012.

[43] Wenchao Li, A. Gascon, P. Subramanyan, Wei Yang Tan, A. Tiwari, S. Malik, N. Shankar, and S.A. Seshia. Wordrev: Finding word-level structures in a sea of bit-level gates. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 67–74, June 2013.

[44] Florian Lonsing and Armin Biere. Depqbf: A dependency-aware qbf solver. *JSAT*, 7 (2-3):71–76, 2010. URL `http://jsat.ewi.tudelft.nl/content/volume7/JSAT7_6_Lonsing.pdf`.

[45] Kathy Macropol. Clustering on graphs: The Markov cluster algorithm (MCl). Presentation, Computer Science, UCSB, 2009.

[46] I. McLoughlin. Secure embedded systems: The threat of reverse engineering. In *Parallel and Distributed Systems, 2008. ICPADS '08. 14th IEEE International Conference on*, pages 729–736, Dec 2008.

[47] Travis Meade, Shaojie Zhang, and Yier Jin. Netlist reverse engineering for high-level functionality reconstruction.

[48] C. Miller, K. Gitina, and B. Becker. Bounded model checking of incomplete real-time systems using quantified smt formulas. In *Microprocessor Test and Verification (MTV), 2011 12th International Workshop on*, pages 22–27, December 2011.

[49] A. Mishchenko, S. Chatterjee, and R.K. Brayton. Improvements to technology mapping for LUT-based FPGAs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):240–253, February 2007. ISSN 0278-0070.

[50] J. Mohnke and S. Malik. Permutation and phase independent boolean comparison. In *Design Automation, 1993, with the European Event in ASIC Design. Proceedings. [4th] European Conference on*, pages 86–92, February 1993.

[51] Janett Mohnke, Paul Molitor, and Sharad Malik. Limits of using signatures for permutation independent boolean comparison. *Formal Methods in System Design*, 21(2):167–191, September 2002. ISSN 1572-8102.

[52] NanGate. Nangate 45nm open cell library, 2015. URL `http://www.nangate.com/`.

[53] M. Oya, Youhua Shi, M. Yanagisawa, and N. Togawa. A score-based classification method for identifying hardware-trojans at gate-level netlists. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pages 465–470, March 2015.

[54] Peichen Pan and Chih-Chang Lin. A new retiming-based technology mapping algorithm for LUT-based FPGAs. In *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, FPGA '98, pages 35–42, New York, NY, USA, 1998. ACM. ISBN 0-89791-978-5.

[55] Darsh P. Ranjan, Daijue Tang, and Sharad Malik. A comparative study of 2qbf algorithms. In *Theory and Applications of Satisfiability Testing*, 2004.

[56] M. Rostami, F. Koushanfar, and R. Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, August 2014. ISSN 0018-9219.

[57] J. Paul Roth. *Computer Logic, Testing and Verification*. W. H. Freeman & Co., New York, NY, USA, 1980. ISBN 0914894625.

[58] N. Rubanov. Subislands: the probabilistic match assignment algorithm for subcircuit recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(1):26–38, Jan 2003. ISSN 0278-0070.

[59] Horst Samulowitz. *Solving Quantified Boolean Formulas*. PhD thesis, University of Toronto, 2007.

[60] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[61] Ulf Schlichtmann, Franc Brglez, and Peter Schneider. Efficent boolean matching based on unique variable ordering. In *International Workshop on Logic Synthesis*, 1993.

[62] Yiqiong Shi, Chan Wai Ting, Bah-Hwee Gwee, and Ye Ren. A highly efficient method for extracting FSMs from flattened gate-level netlist. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 2610–2613, May 2010.

[63] Yiqiong Shi, Bah-Hwee Gwee, Ye Ren, Thet Khaing Phone, and Chan Wai Ting. Extracting functional modules from flattened gate-level netlist. In *Communications and Information Technologies (ISCIT), 2012 International Symposium on*, pages 538–543, October 2012.

[64] Hong-Yan Su, Chih-Hao Hsu, and Yih-Lang Li. Subhunter: A high-performance and scalable sub-circuit recognition method with prüfer-encoding. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pages 1583–1586, March 2015.

[65] P. Subramanyan, N. Tsiskaridze, Wenchao Li, A. Gascon, Wei Yang Tan, A. Tiwari, N. Shankar, S.A. Seshia, and S. Malik. Reverse engineering digital circuits using structural and functional analyses. *Emerging Topics in Computing, IEEE Transactions on*, 2(1):63–80, March 2014. ISSN 2168-6750.

[66] P. Subramanyan, S. Ray, and S. Malik. Evaluating the security of logic encryption algorithms. In *Hardware Oriented Security and Trust (HOST), 2015 IEEE International Symposium on*, pages 137–143, May 2015.

[67] Pramod Subramanyan, Nestan Tsiskaridze, Kanika Pasricha, Dillon Reisman, Adriana Susnea, and Sharad Malik. Reverse engineering digital circuits using functional analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1277–1280, March 2013.

[68] Randy Torrance and Dick James. The state-of-the-art in ic reverse engineering. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 363–381, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-04137-2. URL `http://dx.doi.org/10.1007/978-3-642-04138-9_26`.

[69] Oklahoma State University. Oklahoma state university cell library (osu035), 2015. URL `http://vlsiarch.ecen.okstate.edu/flow/`.

[70] Stijn van Dongen. MCl-edge. URL `http://micans.org/mcl/`.

[71] Stijn van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, May 2000. URL `http://www.library.uu.nl/digiarchief/dip/diss/1895620/inhoud.htm`.

[72] Stijn van Dongen. A cluster algorithm for graphs. Technical report, National Research Institute for Mathematics and Computer Science in the Netherlands, 2000. URL `http://www.cwi.nl/ftp/CWIreports/INS/INS-R0010.ps.Z`.

[73] Y. Vizel, G. Weissenbacher, and S. Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, November 2015. ISSN 0018-9219.

[74] Scott West. Sat and smt solver basics. Lecture ETH Zürich, Novemeber 2011.

[75] Jennifer L. White, Anthony S. Wojcik, Moon-Jung Chung, and Travis E. Doom. Candidate subcircuits for functional module identification in logic circuits. In *Proceedings of the 10th Great Lakes Symposium on VLSI*, GLSVLSI '00, pages 34–38, New York, NY, USA, 2000. ACM. ISBN 1-58113-251-4.

[76] J.L. White, M.J. Chung, A.S. Wojcik, and T.E. Doom. Efficient algorithms for subcircuit enumeration and classification for the module identification problem. In *Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on*, pages 519–522, 2001.

[77] Clifford Wolf. Yosys open synthesis suite. `http://www.clifford.at/yosys/`. URL `http://www.clifford.at/yosys/`.

[78] Tony F Wu, Karthik Ganesan, Alexander Hu, H-S Philip Wong, Simon Wong, and Subhasish Mitra. Tpad: Hardware trojan prevention and detection for trusted integrated circuits. *arXiv preprint arXiv:1505.02211*, 2015.

[79] Jin-Tai Yan and Pei-Yung Hsiao. A new fuzzy-clustering-based approach for two-way circuit partitioning. In *VLSI Design, 1995., Proceedings of the 8th International Conference on*, pages 359–364, January 1995.